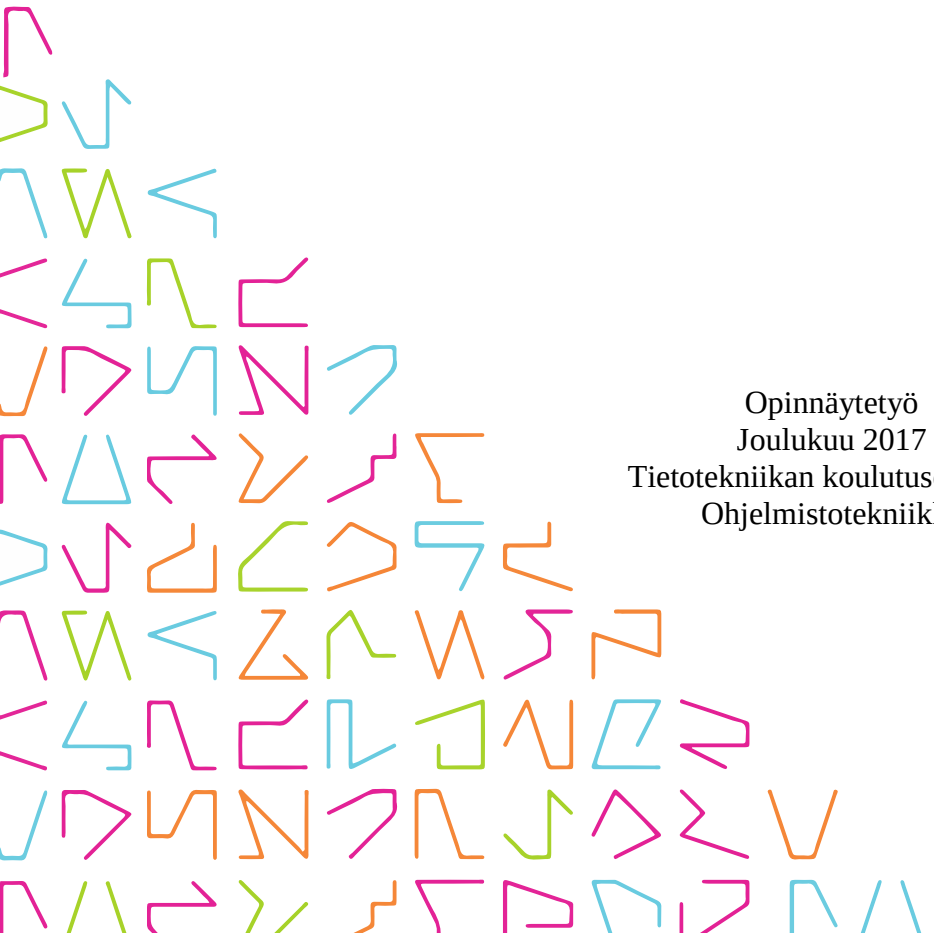
TAMPEREEN
AMMATTIKORKEAKOULU

KUBERNETES-KLUSTERIN ASENNUS JA KÄYTTÖÖNOTTO

Reko Peltola



Opinnäytetyö
Joulukuu 2017
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

PELTOLA, REKO:
Kubernetes-klusterin asennus ja käyttöönotto

Opinnäytetyö 57 sivua, joista liitteitä 16 sivua
Joulukuu 2017

Tässä opinnäytetyössä tutkittiin avoimen koodin Docker- ja Kubernetes-teknologioita. Teknologioilla tavoitellaan nykyaikaisempaa ja helpommin ylläpidettävää infrastruktuuria ohjelmistoprojekteille. Teknologiat mahdollistavat lisäksi helpon siirtymän eri infrastruktuurien, kuten pilvipalveluiden ja oman palvelinkeskuksen välillä. Työn tuloksia voidaan käyttää asennusohjeena vastaavan ympäristön asennukseen ja käyttöönottoon.

Työssä kuvataan Docker- ja Kubernetes-teknologioiden keskeiset ominaisuudet ja tehtiin kehitystyö, jossa Kubernetes-klusteri asennettiin ja otettiin rajoitettuun testikäyttöön kahdelle erityyppiselle projektille Power Finland Oy:llä. Sovellukselle tehtiin konfiguraatiot, joilla ne saatiin paketoitua Docker-muotoon. Lisäksi sovelluksia varten tehtiin konfiguraatiot Kuberneteksessä ajoa varten. Tehtyjä konfiguraatioita voidaan hyödyntää myöhemmin vastaavanlaisten projektien siirtämisessä Kubernetekseen.

Opinnäytetyössä asennettua ja käyttöönotettua Kubernetes-klusteria voidaan jatkokehittää muun muassa tuomalla siihen tuki Docker for Windows -teknologialle, jonka avulla Kubernetekseen voidaan asentaa projekteja, jotka vaativat Windows-käyttöjärjestelmän. Laajemmassa käyttöönotossa olisi myös hyvä parantaa tunnistautumista ja kahdentaa Kuberneteksen sisäisesti tarvitsemia komponentteja.

Asiasanat: kubernetes, docker, arkkitehtuuri, infrastruktuuri

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software engineering

PELTOLA, REKO:
Installation and deployment of Kubernetes-cluster

Bachelor's thesis 57 pages, appendices 16 pages
December 2017

This thesis examines open source Docker and Kubernetes technologies. These technologies can be used to modernize application deployment environments and making it easier to switch between different infrastructures, such as public cloud environments or private datacenters. The results of this thesis can be used as basis for installation and deployment of similar setups.

This thesis describes the key principals of those technologies. As a part of the project a Kubernetes cluster was installed and deployed as a proof of concept for Power Finland Oy. Two different software projects were “dockerized” and deployed to Kubernetes. The Docker and Kubernetes configurations made in this project can be used to dockerize and move other similar projects to Kubernetes.

The Kubernetes cluster installed and deployed in this project can be further improved by adding support for Windows based projects. This can be achieved with the Docker for Windows technology. Authentication and authorization can also be improved as well as redundancy added to internal Kubernetes components.

Key words: kubernetes, docker, architecture, infrastructure

SISÄLLYS

LYHENTEET JA TERMIT

Alpine	Linux jakelu
ASP.NET Core	Microsoftin kehittämä sovelluskehys
CentOS	Linux jakelu
Container	Suomeksi kontti, docker-imagesta käynnistetty instanssi
Dockerfile	Docker-imagen automaattiseen rakentamiseen käytettävä tiedosto, joka kuvaa tarvittavat toimenpiteet
Kubernetes Master-Node	Kuberneteksen pääpalvelin, joka on vastuussa klusterin hallinnasta
Kubernetes Worker-Node	Kuberneteksen palvelin, jossa työkuormaa ajetaan
NAT	Network Address Translation
NFS	Network File System, verkkotiedostojärjestelmä
Node	Klusteriin kuuluva palvelin
Ubuntu	Linux jakelu
WSL	Windows Subsystem for Linux

1 JOHDANTO

Power Finland Oy on elektroniikan vähittäismyyntiin erikoistunut yritys, joka operoi verkkokauppaa ja myymälöitä Suomessa. Se on osa Power International AS -konsernia, joka omistaa Expert- ja Power-liiketjut Suomessa, Ruotsissa, Tanskassa ja Norjassa. Konsernin liikevaihto on noin 900 miljoonaa euroa.

Powerin kehitystiimi, joka toimii Suomesta käsin, kehittää ja ylläpitää useita erilaisia ohjelmistoprojekteja. Ohjelmistot ovat pääasiassa verkkosivustoja, kuten yhdistetty Suomen, Ruotsin, Norjan ja Tanskan verkkokauppa-alusta tai erilaiset sisäiseen käyttöön tarkoitetut selainpohjaiset sovellukset. Lisäksi käytössä on erilaisia tiedon prosessointiin tarkoitettuja sovelluksia.

Käyttöönottovaiheessa ohjelmistot on siirretty eri palvelimille, ja esimerkiksi ajastettujen tehtävien hallintaan on käytetty Windowsin sisäänrakennettua toiminnallisuutta. Ajan mittaan ohjelmistoja on hajaantunut useille eri palvelimille ja niiden päivittäminen käsin on työläs ja virhealtis prosessi.

Uudemmat projektit on toteutettu ASP.NET Core -teknologialla, joten ne toimivat kaikilla käyttöjärjestelmillä ja ovat helposti paketoitavissa Dockerin tukemaan muotoon. Docker yhdessä Kuberneteksen kanssa vaikuttaa sopivalta yhdistelmältä ja siihen on myös mahdollista saada tuki Windows-käyttöjärjestelmälle Powerin vanhempia projekteja silmällä pitäen.

Kubernetes-alustalta tavoitellaan pääasiassa ohjelmistojen luotettavuuden parantamista, päivitysten parempaa hallittavuutta ja mikropalveluarkkitehtuurin mahdollistamista tuomalla siihen tarvittavat työkalut. Lisäksi Kubernetes toimii niin pilvipalveluissa kuin omassa palvelinkeskuksessakin, eli se mahdollistaa myöhemmin pilvipalveluihin siirtymisen ilman muutoksia sovelluksiin. Lisäksi Kubernetes poistaa tavallisilta kehittäjiltä tarpeen päästä käsiksi virtuaalipalvelimiin, koska kaikki Kuberneteksen hallinta tapahtuu sen rajapintaa käyttämällä.

Tässä opinnäytetyössä kuvataan kehitystyö, jonka tarkoituksena on pystyttää ja käyttöönottaa avoimen koodin Kubernetes-alusta yhdessä avoimen Docker-teknologian kanssa.

Kubernetes otetaan aluksi rajoitettuun koekäyttöön muutamille ASP.NET Core -pohjaisille projekteille, joita tullaan ajamaan Linux-ympäristössä. Asennus ja käyttöönotto dokumentoidaan myöhempää käyttöä varten, kun Kubernetes otetaan laajempaan tuotantokäyttöön.

2 DOCKER

2.1 Docker teknologiana

Docker on alusta, joka tarjoaa työkalut ohjelmien ja niiden riippuvuuksien paketointiin. Tätä luotua pakettia kutsutaan imageksi ja imagesta Dockerilla käynnistettyä instanssia containeriksi. Paketoitu image sisältää valitun käyttöjärjestelmän, itse ohjelmiston ja sen mahdolliset riippuvuudet, esimerkiksi tietty versio käyttöjärjestelmään asennettavasta paketista. Imagesta voidaan käynnistää container missä tahansa käyttöjärjestelmässä tai pilvipalveluissa, jotka vain tukevat Docker-teknologiaa. Tämän ansioista containerit ovat helposti siirrettävä teknologia. (Docker overview 2017.)

Linuxin osalta käyttöjärjestelmä voi esimerkiksi olla tavallinen Ubuntu tai CentOS, mutta myös erittäin pienikokoinen Alpine. Alpinen ladattava koko on vain noin 5 MB, kun taas Ubuntu on kooltaan noin 129 MB ja CentOS noin 192 MB. (Glider Labs 2017.)

Käyttöjärjestelmä voi myös olla Windows Server Core tai Windows Nano Server. Ne vaativat kuitenkin toimiakseen Dockerin Windows-version, joka on rajattu tämän työn ulkopuolelle. (Docker For Windows Server 2017.)

Docker containereita ei virtualisoida kuten tavallisia virtuaalikoneita, vaan ne ajetaan suoraan Dockeria suorittavan koneen kernelissä. Näin Docker säästää resursseja, ja samoille palvelimille mahtuu enemmän hyötykuormaa. (Docker overview 2017.)

2.2 Dockerfile

Docker image rakennetaan joko käsin tai automaattisesti Dockerfileen määritettyjen komentojen mukaan. Dockerfile on tekstitiedosto, jonka `docker build` -komento lukee ja muodostaa Docker imagen siihen määriteltyjen komentojen mukaan. (Dockerfile reference 2017.)

Kuvassa 1 on esimerkki yksinkertaisesta Dockerfilestä. Siinä otetaan imagen pohjaksi Ubuntu 15.04:sta jo valmiiksi luotu container, joka on saatavilla Docker Hub -nimisessä

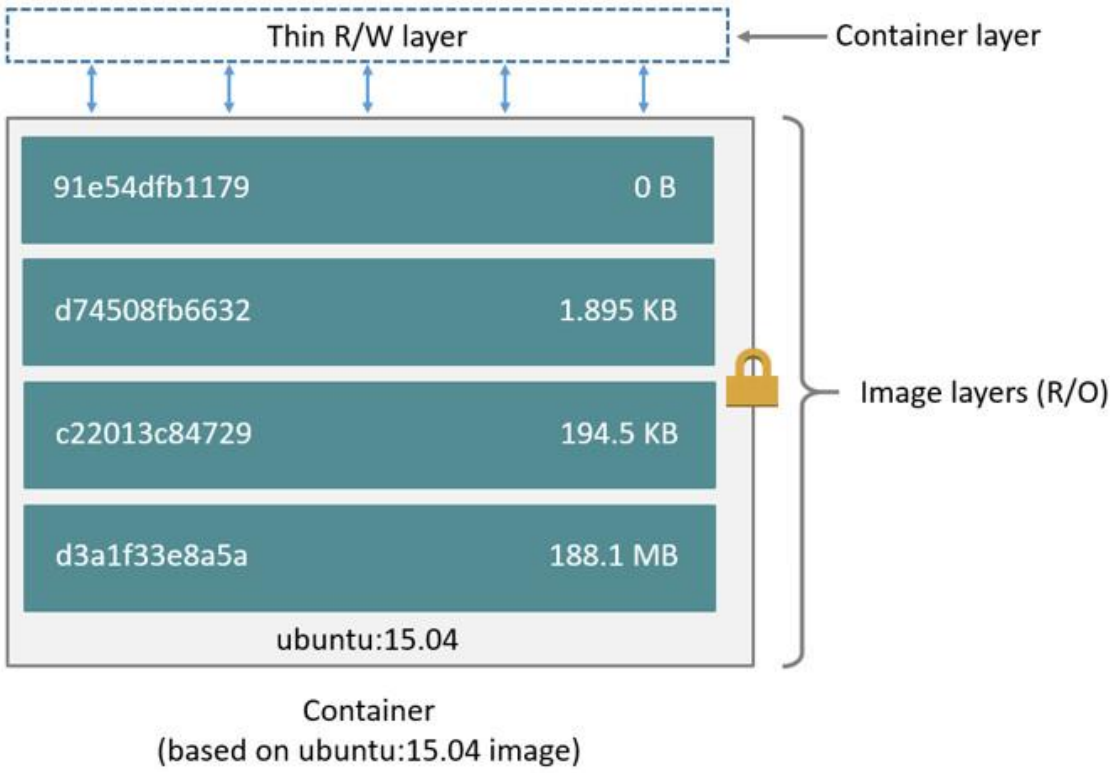
palvelussa. Tähän valmiiseen käyttöjärjestelmä-imageen kopioidaan COPY-komennolla kansio ja sen tiedostot komennon suorittavalta koneelta. Seuraavaksi suoritetaan imagessa make-komento, joka kääntää ohjelman containerin sisällä. Lopuksi CMD-komento asettaa imagelle oletusarvoisen komennon, joka suoritetaan, kun container käynnistetään. (Dockerfile reference 2017.)

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

KUVA 1. Yksinkertainen Dockerfile (About images, containers, and storage drivers 2017.)

2.3 Docker image

Docker image koostuu useista eri kerroksista eli layereistä. Jokainen layer muodostuu yhdestä Dockerfilen komennosta ja nämä layerit eivät ole muokattavissa luomisen jälkeen. Kaavio 1 kuvaa Docker imagen rakenteen, kun sen rakentamiseen käytettiin kuvan 1 mukaista Dockerfilea. Kaaviota luetaan alhaalta ylöspäin, eli ensimmäinen layer muodostuu Dockerfilen ensimmäisestä komennosta. (About images, containers, and storage drivers 2017.)



KAAVIO 1. Ubuntu-pohjaisen containerin rakenne (About images, containers, and storage drivers 2017.)

Käynnistyksen yhteydessä containeriin liitetään niin sanottu container layer. Kaikki containerin tekemät muutokset tapahtuvat tähän kerrokseen. Näin samasta imagesta voidaan käynnistää monta eri kopiota resursseja haaskaamatta, kun jokaiselle kopiolle tarvitaan vain oma kerros muutoksia varten. (About images, containers, and storage drivers 2017.)

2.4 Docker registry

Docker registry on palvelinohjelmisto, jonka välityksellä Docker imageita siirrellään tai julkaistaan. Docker-teknologian takana oleva yhtiö tarjoaa registryä Docker Hub -nimisenä palveluna. Vaihtoehtoisesti tarjolla on myös Docker image, jolla rekisteri voidaan asentaa omaan ympäristöön. (Docker Registry 2017.)

Docker registry hyödyntää Docker imagen kerroksia ja siirtää verkon yli vain ne osat, jotka eivät jo ole rekisterissä tai käyttäjän koneella. Imageita on myös mahdollista siirtää `docker save` ja `docker load` komennoilla, mutta näillä komennoilla kerroksia ei voida hyödyntää ja resursseja käytetään tarpeettomasti.

2.5 Dockerin hyödyt

Docker tarjoaa avoimen koodin teknologian ja työkalut, jolla ohjelmat saadaan muutettua tavanomaisista vaikeasti siirrettävistä virtuaalikoneista container-muotoon. Containerit ovat kevyempiä, helpommin siirrettävissä olevia ja niitä voidaan ajaa kaikkialla missä Docker on tuettu. Suurimmat pilvipalvelut kuten Amazon Web Services, Google Compute Engine ja Azure tarjoavat Dockeria suoraan tai Kuberneteksen välityksellä palveluna (Azure Container Service (AKS) 2017; Docker Basics - Amazon Elastic Container Service 2017; Kubernetes engine 2017). Lisäksi missä tahansa pilvipalvelussa voidaan ajaa Dockeria omassa virtuaalikoneessa.

Dockerin työkalut sopivat erittäin hyvin myös yhteen automaation kanssa ja se on helppo ottaa mukaan jo olemassa oleviin ratkaisuihin. Docker-containerit ovat kevyitä ja eivät juurikaan haaskaa resursseja. Ne ovat hyvä pohja mikropalveluarkkitehtuurille, jossa tavanomaisia isoja ohjelmistoja lähdetään jakamaan pienempiin, helpommin ja nopeammin ylläpidettäviin palasiin.

3 KUBERNETES

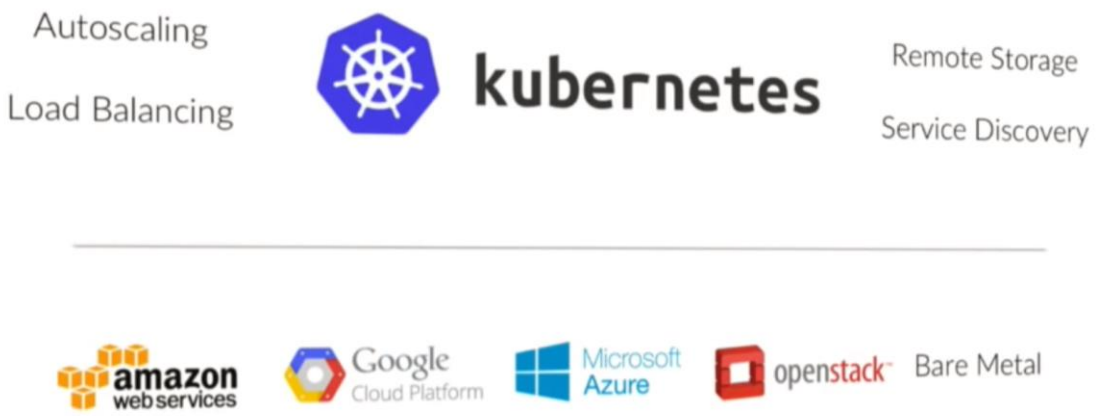
3.1 Kubernetes alustana

Yhtä pilvipalvelua vastaan kehitetyn sovelluksen siirtäminen esimerkiksi toiseen pilvipalveluun tai käyttöönotto asiakkaan ympäristöön voi olla työläs ja aikaa vievä prosessi. Se ei välttämättä onnistu ollenkaan, jos sovellus hyödyntää runsaasti pilvipalvelun ominaisuuksia, kuten kuormantasausta tai kuorman mukaan automaattisesti säätyviä resursseja. Kuvassa 2 on listattuna muutaman pilvipalvelun eri käyttötarkoituksiin tarjoamat teknologiat. Pilvipalveluita verrataan kuvassa myös pilven ulkopuoliseen omaan infrastruktuuriin, jota kutsutaan Bare Metal -termillä.

	amazon web services	Google Cloud Platform	Microsoft Azure	openstack	Bare Metal
Autoscaling	Autoscaling Group	Autoscaler	Scale Set	Heat Scaling Policy	X
Load Balancing	Elastic Load Balancer	Load Balancer	Load Balancer	LBaaS	X
Remote Storage	Elastic Block Store	Persistent Disk	File Storage	Block Storage	X
Service Discovery	X	X	X	X	X

KUVA 2. Kuvakaappaus pilvipalveluiden ja fyysisen raudan tukemista ominaisuuksista (Kubernetes: Finally...A True Cloud Platform, Youtube 2016).

Sam Ghods nostaa esityksessään Kuberneteksen vastauksena yhteensopivuusongelmiin, joita eri pilvipalveluiden sekä oman infrastruktuuriin välillä on. Yhteensopivuus mahdollistuu, kun Kubernetes nostetaan kuvan 3 mukaisesti abstraktioksi eri alustojen päälle. Kubernetes hoitaa kaikki sovelluksien tarvitsemat palvelut, kuten kuormantasauksen joko itse tai hyödyntäen taustalla olevaa pilvipalvelua. (Kubernetes: Finally...A True Cloud Platform 2016.)



KUVA 3. Kuvakaappaus, jossa Kubernetes kuvataan pilvipalveluita yhdistäväksi alustaksi (Kubernetes: Finally...A True Cloud Platform, Youtube 2016).

3.2 Verkko (flannel)

Kubernetes itsessään ei pidä sisällään mitään verkkorajapintaa, vaan ne asennetaan liitännäisenä. Kubernetes asettaa verkolle muutamat vaatimukset, jotka tulee toteutua liitännäisten toimesta:

- Kaikkien containereiden tulee voida kommunikoida keskenään ilman NAT:ia
- Kaikkien nodejen tulee voida kommunikoida containereiden kanssa ilman NAT:ia
- Containerin itsensä näkemä IP-osoite tulee olla sama, kuin muut näkevät

Yksinkertainen verkkorajapinta, joka täyttää Kuberneteksen vaatimukset, on flannel, jota tässä projektissa käytettiin. Flannel toimii yhdessä Kuberneteksen rajapinnan kanssa ja jakaa jokaiselle Kuberneteksen palvelimelle yksilöllisen IP-osoiteavaruuden (Cluster Networking 2017; Flannel is a network fabric for containers 2017).

3.3 Kirjautuminen ja valtuutus

Kuberneteksessä on kahdenlaisia käyttäjiä: normaaleja sekä Kuberneteksen hallinnoimia palvelutunnuksia. Palvelutunnukset kuuluvat tiettyihin nimiavaruuksiin ja ne luodaan joko automaattisesti tai Kuberneteksen rajapintaa käyttämällä. (Authenticating 2017.)

Normaaleja käyttäjätunnuksia Kubernetes ei hallinnoi vaan ne tulee hallita ulkoisissa järjestelmissä. Kubernetekseen ne otetaan käyttöön esimerkiksi valtuuttaja-välityspalvelimella. Välityspalvelimen tehtävä on hoitaa kirjautuminen ja valtuutus ja lisätä näistä saadut tiedot HTTP-protokollan tunnisteisiin, josta Kubernetes lukee ne. Yksinkertaisempi, mutta vaikeammin hallittava tapa ovat käyttäjäkohtaiset staattiset tokenit. Tokeneita varten luodaan tiedosto Kuberneteksen master-nodelle, ja API-palvelimen konfiguraatioon laitetaan viittaus tähän tiedostoon. (Authenticating 2017.)

3.4 Kuberneteksen Containerit ja Podit

Kuberneteksessä container tarkoittaa yksittäistä Docker imagesta käynnistettyä instanssia. Dockerin tilalla voidaan käyttää myös jotain muuta Kubernetes yhteensopivaa container-ajoympäristöä kuten Dockerin pohjalla toimivaa containerd-teknologiaa. (Containerd Brings More Container Runtime Options for Kubernetes 2017.)

Pod taas on pienin yksittäinen Kuberneteksen rakennuspalikka. Pod kapseloi yhden tai useamman containerin, niiden tarvitseman tallennustilan, yhteisen uniikin IP-osoitteen ja tiedot miten containerin ajoon liittyvät asetukset. (Pods 2017.)

Yksi container podia kohti on yleisin Kuberneteksessä käytettävä malli. Siinä pod toimii pelkkänä kääreenä yksittäiselle containerille. Useampi container podia kohti malli on vähemmän käytetty, mutta sillä voidaan toteuttaa edistyneempiä ja monimutkaisempia malleja. (Pods 2017.)

Podit ovat luonteeltaan kertakäyttöisiä ja lyhytaikaisia. Kun Kubernetes tai käyttäjä luo podin, se ajastetaan jollekin worker-nodelle. Pod pysyy worker-nodella kunnes sen sisältämä prosessi sammuu, pod poistetaan, evakuoidaan resurssien puutteen vuoksi tai kunnes worker-node jostain syystä lakkaa toimimasta. (Pods 2017.)

3.5 Controller

Podien luonteen vuoksi käyttäjät harvoin hallitsevat niitä suoraan, sen sijaan käytetään korkeamman abstraktion komponentteja, controllereita. Kuberneteksessä on monia erilaisia controllereita, jotka on tarkoitettu erilaisiin käyttötarkoituksiin.

Replica Sets sekä Replication Controller abstraktiot huolehtivat, että määriteltyjä podeja on aina käynnissä oikea määrä. Eli podeja joko käynnistetään tai sammutetaan, jotta päästää oikeaan asetettuun määrän. Vaikka podeja tarvittaisiin vain yksi, on replication controller silti hyödyllinen. Se varmistaa podien uudelleen luonnin häiriöiden jälkeen, jonka esimerkiksi klusterin noden käyttöjärjestelmän päivitys aiheuttaa. (Replication Controller 2017.)

Cron Jobs -abstraktio tarjoaa mahdollisuuden ajastettuihin tehtäviin Kuberneteksessä. Dokumentaation mukaan on mahdollista, että välillä ajastettu tehtävä käynnistyykin useamman kerran. Sen pitäisi olla harvinaista, mutta käyttäjän ajastetut tehtävät tulee olla suunniteltu tämä huomioiden. (Cron Jobs 2017.)

Deployment-abstraktio huolehtii automaattisesti podien ja replica settien päivityksen määritetyn konfiguraation perusteella. Deploymentin avulla voidaan esimerkiksi hoitaa sovelluksen hallittu päivitys. Deployment käynnistää taustalla uudet podit ja kääntää liikenteen uuteen versioon. Päivitys voidaan myös keskeyttää helposti ja sovellus vaihtaa takaisin vanhaan versioon, jos esimerkiksi sovelluksen virheiden määrä nousee.

3.6 Service

Kuberneteksessä podit eivät ole kovin pitkäikäisiä tai pysyviä, joten niihin viittaaminen suoraa IP-osoitteilla vaatisi jatkuvaa konfiguraation päivitystä. Controllerien avulla on myös mahdollista luoda useita kopioita podeista, jolloin hallittavana olisi useampi IP-osoite ja sovelluksen tulisi itse jakaa liikenne niiden välillä.

Kuberneteksen service-abstraktio hoitaa automaattisesti taustalla olevien podien linkityksen yhteen luomalla palvelulle klusterin sisällä uniikin IP-osoitteen. Tämä IP-osoite ohjaa

palvelun yhteen tai useampaan ajossa olevaan podiin. Palvelun IP-osoite saadaan ympäristömuuttujista tai kube-dns -lisäosan avulla klusterin nimipalvelimesta. (Services 2017.)

Kuberneteksessä on neljä erilaista tyyppiä palvelulle. ClusterIP-tyyppinen palvelu on saavutettavissa uniikilla IP-osoitteella vain klusterin sisältä käsin. NodePort-tyyppinen palvelu on saavutettavissa klusterin ulkopuolelta jokaisen noden IP-osoitteella ja Kubernetes valitsemalla satunnaisella portilla. LoadBalancer-tyyppinen palvelu on saatavilla vain muutamissa pilvipalveluissa, joissa pilvipalveluiden kuormantasaajat konfiguroitaisiin automaattisesti palvelulle. ExternalName-tyyppinen palvelu lisää palvelulle määritetyn verkkotunnuksen klusterin DNS-palveluun. (Services 2017.)

3.7 Volume

Kuberneteksessä containerin levylle kirjottamat tiedostot säilyvät vain sen aikaa, kun container on käynnissä. Jos container esimerkiksi kaatuu ja Kubernetes joutuu käynnistämään sen uudelleen, niin kaikki containerin kirjoittamat tiedostot katoavat. Usein myös yhdeksi podiksi linkitettyjen containerin tulee voida jakaa tiedostoja keskenään. (Volumes 2017.)

Containereille saadaan pysyvä tallennustila liittämällä siihen Kubernetes tarjoama volume. Volume on Kuberneteksessä abstraktio tallennustilalle ja sille löytyy useita eri toteutuksia, jotka toimivat hieman eri tavalla. Useimmat hajautetut toteutukset, esimerkiksi NFS ja Amazonin Elastic Block Store säilyttävät kaiken tiedon, vaikka pod, mihin tallennustila on liitetty, poistettaisiin. (Volumes 2017.)

Vaikka, toteutuksesta riippuen, volume voi säilyttää tiedon podin elinkaarta pidemmälle, on Kuberneteksessä oma abstraktio pysyvälle tallennustilalle nimeltään persistent volume. Pysyvä tallennustila on joko ylläpitäjän valmiiksi luoma yksittäinen volume tai dynaamisesti luotua, jota pyydetään Kubernetekseltä persistent volume claim -nimisellä konseptilla. Dynaamisessa mallissa ylläpitäjä on asettanut Kuberneteksen luomaan tallennustilaa automaattisesti esimerkiksi NFS-palvelimelle, kun käyttäjät sitä pyytävät.

4 KUBERNETES-KLUSTERIN ASENNUS

4.1 Projektikone

Projekti suoritettiin työasemalta, jossa käyttöjärjestelmänä oli Windows 10 Pro. Projektikoneeseen oli lisäksi asennettu Windows Subsystem for Linux -lisäosa (WSL), jossa Windows-käyttöjärjestelmän sisällä pyöri Ubuntun tekstiversio (Windows 10 Installation Guide 2017).

Projektikoneen WSL-ympäristön SSH-komentoa käytettiin palvelimien hallintaan. Myöhemmin projektikoneelle asennettiin Kuberneteksen hallintaan tarvittava työkalu, jonka avulla Kuberneteksen ohjauspaneeliin saatiin yhteys.

4.2 Palvelimien valmistelu

Projektia varten asennettiin neljä CentOS 7 -käyttöjärjestelmällä varustettua Linux-palvelinta, joista yksi varattiin Kuberneteksen master-nodeksi, kaksi Kuberneteksen worker-nodeiksi ja yksi välityspalvelimeksi. Kuberneteksessä pyöriviä palveluita käytettiin tämän välityspalvelimen kautta, ja se tarjosi palveluille myös HTTPS tuen.

Kaikille palvelimille asennettiin uusimmat päivitykset ja joukko erilaisia apuohjelmia, kuten tekstinmuokkausohjelma vim, oheisilla komennoilla:

```
$ sudo yum makecache && sudo yum -y upgrade
$ sudo yum -y install bash-completion epel-release vim screen dstat
$ sudo yum -y install htop yum-utils wget nfs-utils
```

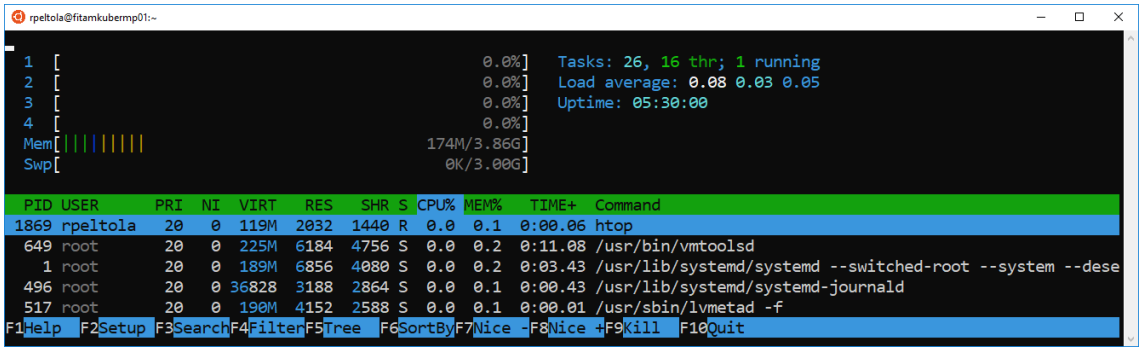
Koska palvelimet ovat sisäverkossa ja palomuuria hallitaan verkkojen välillä keskitetysti, otettiin palomuuri pois käytöstä oheisella komennolla:

```
$ sudo systemctl disable firewalld && sudo systemctl stop firewalld
```

4.3 Kubeadm-työkalun asennus

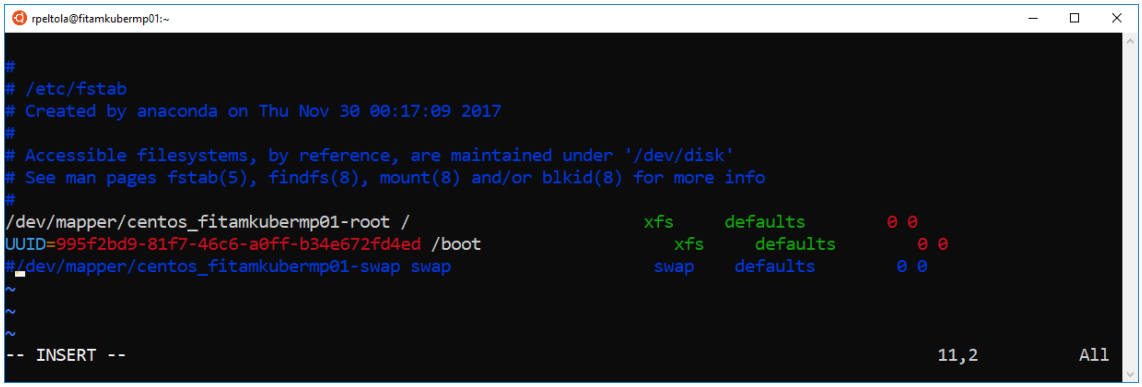
Kuberneteksen dokumentaation mukaan (Installing kubeadm 2017) kubeadm-työkalun vaatimukset jokaiselle palvelimelle ovat:

- Vähintään 2 GB muistia
- Vähintään 2 CPU-ydintä
- Kaikki asennettavat palvelimet voivat kommunikoida keskenään verkossa
- Yksilöllinen MAC-osoite
- Sivutustiedosto, eli Linuxin tapauksessa swap, on pois päältä
- SELinux on poistettu käytöstä



KUVA 4. Htop-työkalu

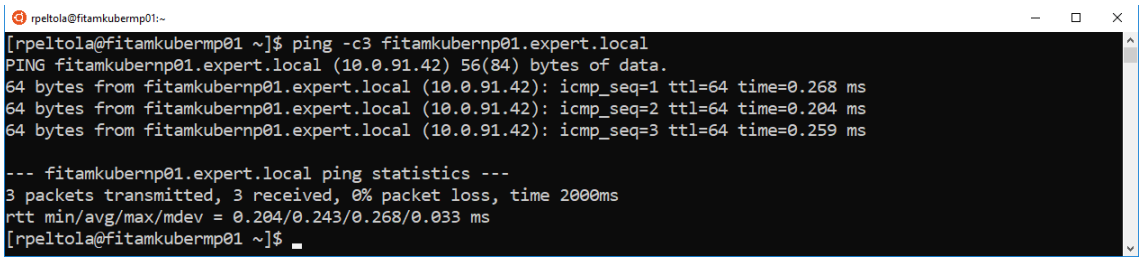
Koneen muistin ja ytimien määrän voi tarkistaa kätevästi visualisella htop-komennolla. Samalla komennolla näkee myös swapin määrän. Komennon esimerkki on kuvassa 4, jossa koneesta oli 3,86 GB käytettävissä olevaa muistia, 4 CPU-ydintä ja 3 GB swapia.



KUVA 5. Swap-osion poistaminen käytöstä lisäämällä # merkki swap-rivin eteen

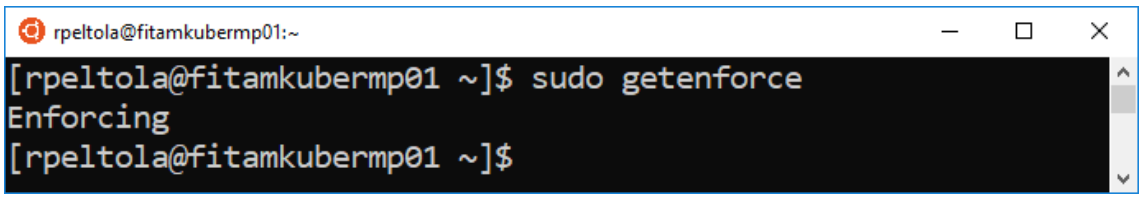
`/etc/fstab` tiedosto kertoo järjestelmälle mitä levyosiota tulee ottaa käyttöön käyttöjärjestelmän käynnistyessä. Muokattiin tiedostoa kuvan 5 näyttämällä tavalla niin, että swap-osiota ei enää otettu käyttöön seuraavan käynnistyksen yhteydessä. Otettiin swap myös pois käytöstä nykyisessä istunnossa käyttämällä oheista komentoa:

```
$ sudo swapoff -a
```



KUVA 6. Palvelinten välisen kommunikaation testaaminen ping-komennolla

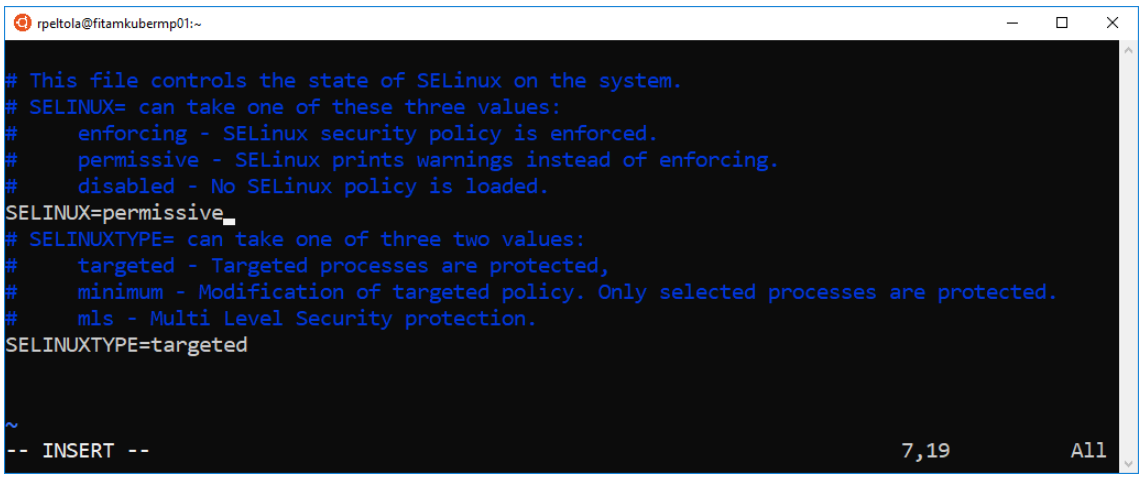
Kuvan 6 mukaisella komennolla testattiin, että palvelimet olivat keskenään saavutettavissa. Komento lähetti kolme ping-käskyä fitamkubermp01 palvelimelta fitamkubernp01:lle ja tulosti lopuksi statistiikat, josta nähtiin, että ping onnistui. Palvelimet pystyivät siis kommunikoimaan keskenään.



KUVA 7. SELinux tilan tarkistamien, tässä se on päällä

Kuvan 7 osoittamalla komennolla tarkistettiin, onko SELinux käytössä vai ei. Tarvittaessa SELinux-toiminnallisuuden voi poistaa käytöstä oheisella komennolla:

```
$ sudo setenforce 0
```



KUVA 8. SELinux poistaminen käytöstä järjestelmän käynnistyksen yhteydessä

Muokattiin lisäksi `/etc/sysconfig/selinux` tiedostoa kuvan 8 näyttämällä tavalla, jotta SELinux ei tullut uudestaan käyttöön järjestelmän seuraavan käynnistyksen yhteydessä.

Kubeadm-työkalun vaatimusten tarkistamisen jälkeen asennetaan nodeille Docker. Kuberneteksen dokumentaation mukaan Docker 17.03 oli uusin Kuberneteksen kanssa toimivaksi todettu versio. Lisättiin Dockerin pakettihakemisto käyttöjärjestelmään oheisella komennolla:

```
$ sudo yum-config-manager -y --add-repo https://down-
load.docker.com/linux/centos/docker-ce.repo
```

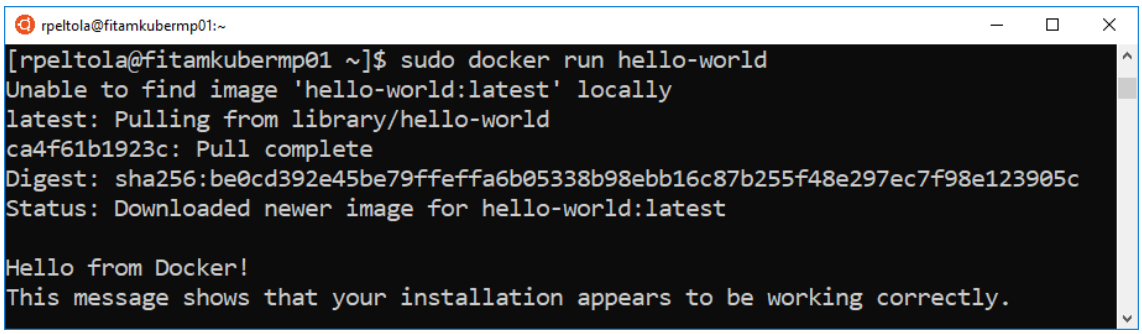
Pakettihakemiston lisäämisen jälkeen Docker 17.03 asennettiin oheisella komennolla:

```
$ sudo yum install --setopt=obsoletes=0 docker-ce-17.03.2.ce-
1.el7.centos.x86_64 docker-ce-selinux-17.03.2.ce-
1.el7.centos.noarch
```

Asennuksen jälkeen käynnistettiin Docker ja asetettiin se myös käynnistymään järjestelmän käynnistyksen yhteydessä oheisella komennolla:

```
$ sudo systemctl start docker && sudo systemctl enable docker
```

Dockerin käynnistyksen jälkeen testattiin kuvan 9 mukaan asentuiko Docker oikein. Onnistuneen asennuksen tulosteen muutama ensimmäinen rivi tulee vastata kuvaa.



KUVA 9. Docker-asennuksen testaaminen

Dockerista käytettiin 17.03 versiota, joka ei ollut uusin pakettihallinnan tarjoama versio. Yhteensopivuusongelmilta välttymiseksi estettiin Dockerin tahaton päivittyminen järjestelmän muiden päivitysten ohella muokkaamalla `/etc/yum.conf` -tiedoston loppuun oheiset rivit:

```
# Exclude docker from upgrade, must specifically upgrade it
exclude=docker*
```

Dockerin asennuksen jälkeen nodeille asennettiin Kuberneteksen hallintaprosessi eli kubelet, Kubernetes klusterin luomiseen tarkoitettu työkalu kubeadm ja Kuberneteksen hallintaan tarkoitettu työkalu kubectl. Kuberneteksen pakettihakemisto lisättiin käyttöjärjestelmään luomalla uusi tiedosto hakemistoon `/etc/yum.repos.d/kubernetes.repo` ja kopioimalla liitteen 1 rivit sinne. Pakettihakemiston lisäämisen jälkeen Kuberneteksen komponentit asennettiin oheisella komennolla:

```
$ sudo yum install -y kubelet kubeadm kubectl
```

Asennuksen jälkeen Kubelet prosessi käynnistettiin ja asetettiin käynnistymään järjestelmän käynnistyksen yhteydessä oheisella komennolla:

```
$ sudo systemctl enable kubelet && sudo systemctl start kubelet
```

Kubelet prosessi jäi odottamaan käskyjä kubeadm-työkalulta, eikä käynnistynyt oikein vielä tässä kohdassa.

Kuberneteksen dokumentaatiossa (Installing kubeadm 2017) varoitetaan, että joillain CentOS 7 käyttäjillä on ollut ongelmia verkkoliikenteen reitityksessä käyttöjärjestelmän

sisällä. Ongelmalta voi välttyä luomalla oheisen konfiguraatiotiedoston `/etc/sysctl.d/kubernetes.conf` ja lisäämällä sinne seuraavat rivit:

```
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

Rivit lisättiin ja muutokset otettiin käyttöön oheisella komennolla:

```
$ sudo sysctl --system
```

Kuberneteksen kubelet-prosessin konfiguraatio eroaa Dockeriin verrattuna, ja se tulee korjata yhteensopivaksi muokkaamalla konfiguraatiota oheisella komennolla:

```
$ sudo systemctl edit kubelet
```

Avautuvaan tekstinmuokkausohjelmaan kopioitiin oheiset rivit, jotka kertoivat kubelet-prosessille oikean ajurin.

```
[Service]
Environment="KUBELET_CGROUP_ARGS=--cgroup-driver=cgroupfs"
```

Palveluun tehdyt muutokset otettiin käyttöön oheisella komennolla:

```
$ sudo systemctl daemon-reload
```

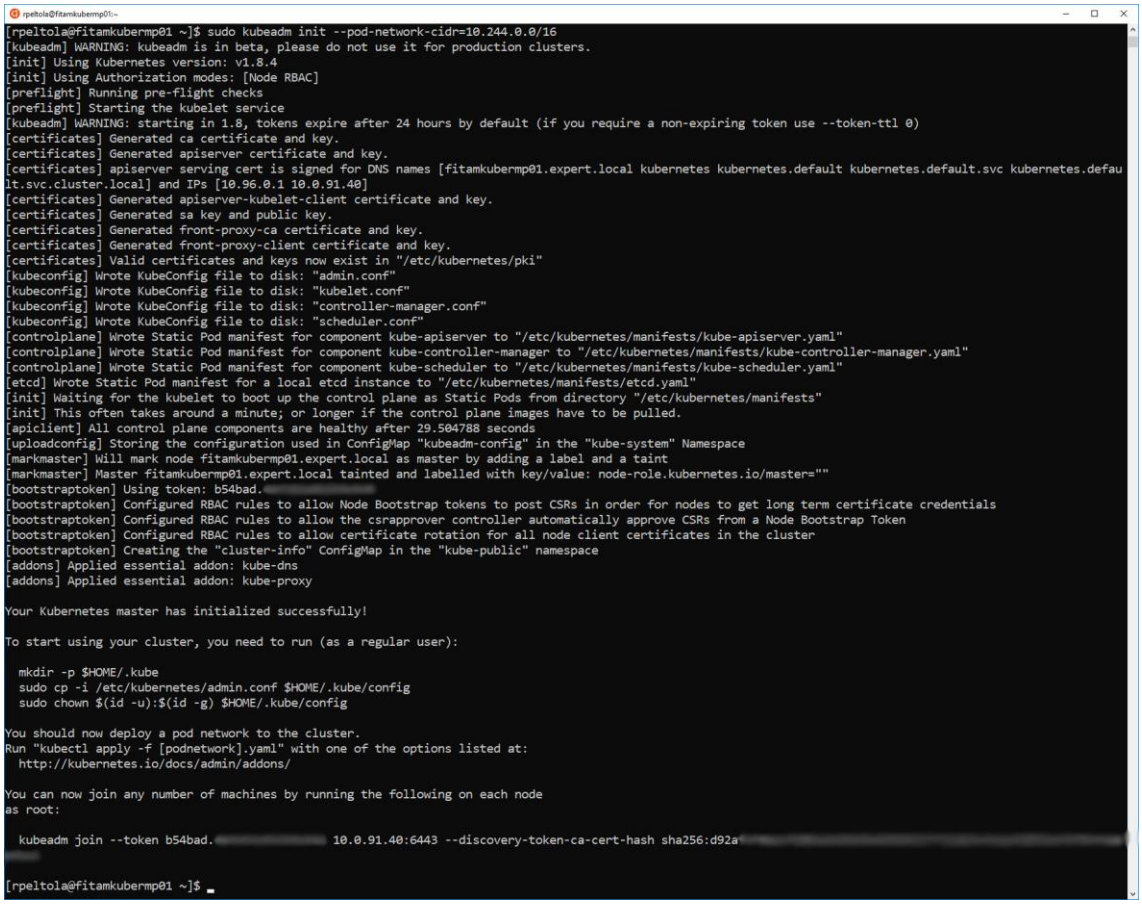
4.4 Kubernetes-klusterin luominen kubeadm-työkalulla

Palvelin nimeltä fitamkubermp01 asennettiin Kubernetes-klusterin master-nodeksi, eli palvelimeksi, joka hallitsee koko Kubernetes-klusteria. Dokumentaation perustella kubeadm-työkalulle tuli antaa parametriksi 10.244.0.0/16 IP-osoiteavaruus, kun käytettiin flannel-verkkomoduulia. Alustettiin klusteri oheisella komennolla:

```
$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

Onnistunut klusterin-alustus on kuvan 10 mukainen. Komennon lopuksi tulostui uusi komento, jolla liitettiin uusia palvelimia luotuun klusteriin, sekä oheinen komento, jolla klusterin hallintaan tarvittava konfiguraatio voidaan kopioida käyttäjän kotihakemistoon.

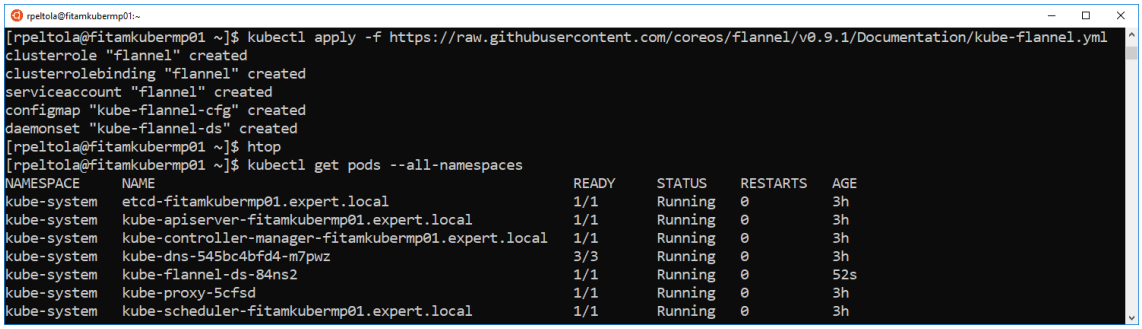
```
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```



KUVA 10. Kubernetes klusterin alustaminen kubeadm-työkalulla

Klusterin alustuksen jälkeen flannel-verkkomoduuli otettiin käyttöön oheisella komennolla:

```
$ kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/v0.9.1/Documentation/kube-flannel.yml
```

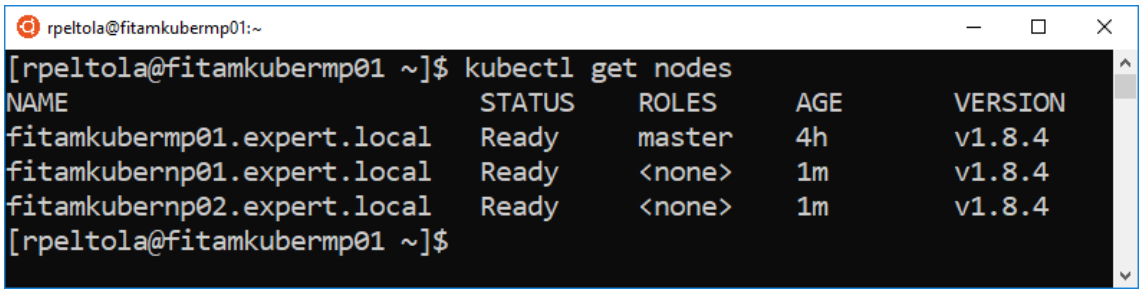


KUVA 11. Flannel-verkkomoduulin lataaminen ja kaikkien podin listaus

Kuva 11 näyttää onnistuneen verkkomoduulin lisäyksen ja listaa kaikki podit klusterissa. Verkkomoduulin lisäys onnistui, kun DNS-podi on käynnistynyt, eli se on tilassa running.

Master-noden alustamisen jälkeen jokaisella worker-nodella ajettiin kubeadm init-komennon aikaisemmin tulostama liittämiskomento, joka oli oheisen kaltainen:

```
$ kubeadm join --token xxx 10.0.91.40:6443 --discovery-token-ca-cert-hash sha256:xxx
```

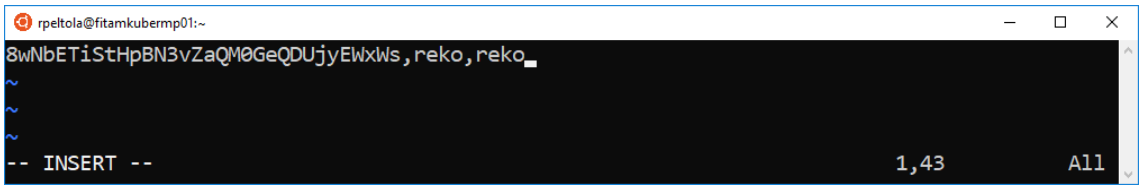


KUVA 12. Klusteriin kuuluvien palvelinten listaaminen pääpalvelimelta

Worker-nodejen lisäämisen jälkeen ajettiin pääpalvelimelta kuvan 12 mukainen komento josta nähtiin, että worker-nodet olivat liittyneet klusteriin onnistuneesti.

4.5 Hallintatunnuksen luominen

Tunnistautumista varten Kuberneteksen master-nodelle luotiin `/etc/kubernetes/static-token.csv` -tiedosto, johon käyttäjäkohtaiset staattiset merkkijonot tallennettiin. Tiedoston tuli olla kuvan 13 mukaisessa formaatissa, jossa rivillä ensimmäisenä oli käyttäjän token, jonka jälkeen käyttäjätunnus ja käyttäjän tunniste (Authenticating 2017). Käyttäjän tunniste voi olla sama kuin itse käyttäjätunnus.



KUVA 13. Esimerkki token-tiedostosta

Kuberneteksen API-palvelimelle tuli antaa kyseinen tiedosto parametrina, jotta tunnistautuminen saatiin toimimaan (Authenticating 2017). Muokattiin master-nodella `/etc/kubernetes/manifests/kube-apiserver.yaml` -tiedostoa lisäämällä konfiguraation command-kohtaan parametri: `--token-auth-file=/etc/kubernetes/static-token.csv` sekä lisäämällä volume ja volumeMount `/etc/kubernetes/static-token.csv` -tiedostolle. Liitteessä 2 on kyseinen konfiguraatiotiedosto kokonaisuudessaan.

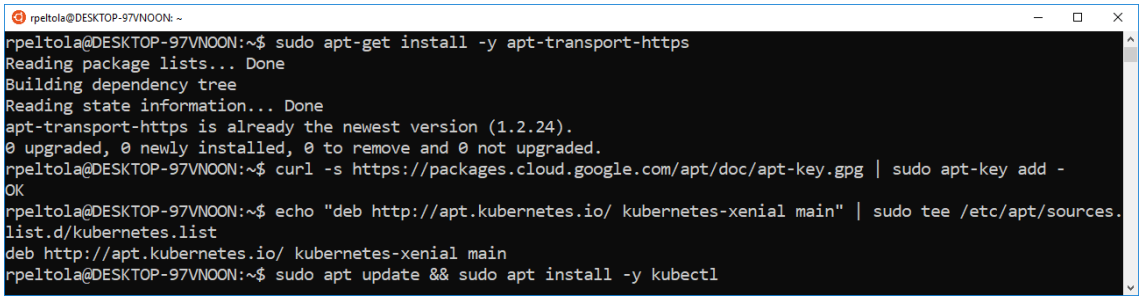
Kuberneteksessä volumeMount tarkoittaa polkua tai tiedostoa containerin sisällä, joka tuodaan sinne containerin ulkopuolelta, jotta tiedostot säilyvät containerin elinkaarta pidemmän ajan. Volumella määritetään mihin tuo containerin sisällä oleva polku oikeasti ohjaa, se voi esimerkiksi olla kansio palvelimella, jolla container on ajossa.

Token-tiedostoon lisätylle käyttäjälle annettiin cluster-admin -niminen rooli. Tällä roolilla käyttäjä sai oikeudet nähdä ja hallita koko klusterin tietoja. (Using RBAC Authorization 2017.)

```
$ kubectl create clusterrolebinding reko-cluster-admin-binding --
clusterrole=cluster-admin --user=reko
```

4.6 Ohjauspaneelin käyttöönotto

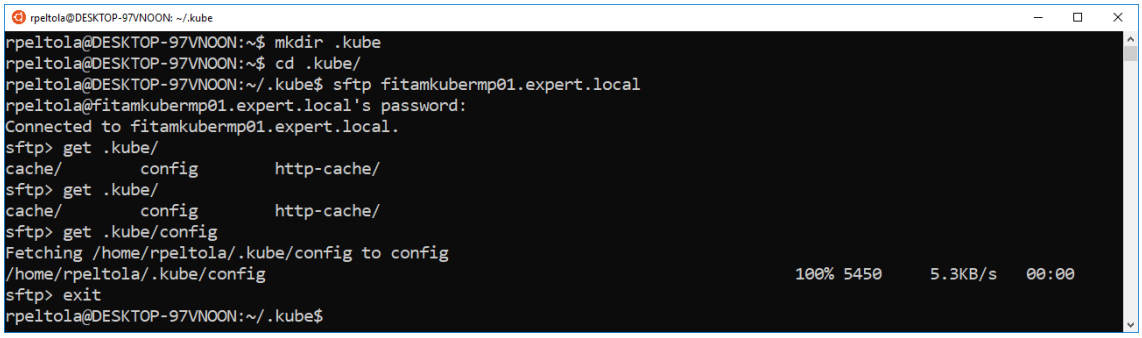
Ohjauspaneelin käyttöönottoa varten projektikoneelta tarvittiin pääsy Kubernetes-klusterin rajapintaan. Asennettiin projektikoneen WSL-ympäristöön kubectl työkalu kuvan 14 mukaisesti.



KUVA 14. Kubectl-työkalun asennus WSL-ympäristöön

Asennuksen jälkeen suoritettiin kuvan 15 mukaiset toimenpiteet, jossa luotiin konfiguraatiokansio projektikoneelle ja haettiin Kuberneteksen master-nodelta asetustiedosto sinne. Tämän jälkeen voitiin testata asennuksen toimivuutta ajamalla projektikoneen WSL-ympäristössä oheinen komento, joka listasi kaikki Kubernetes-klusterissa olleet podit:

```
$ kubectl get pods --all-namespaces
```



KUVA 15. Kubectl-työkalun konfigurointi WSL-ympäristöön

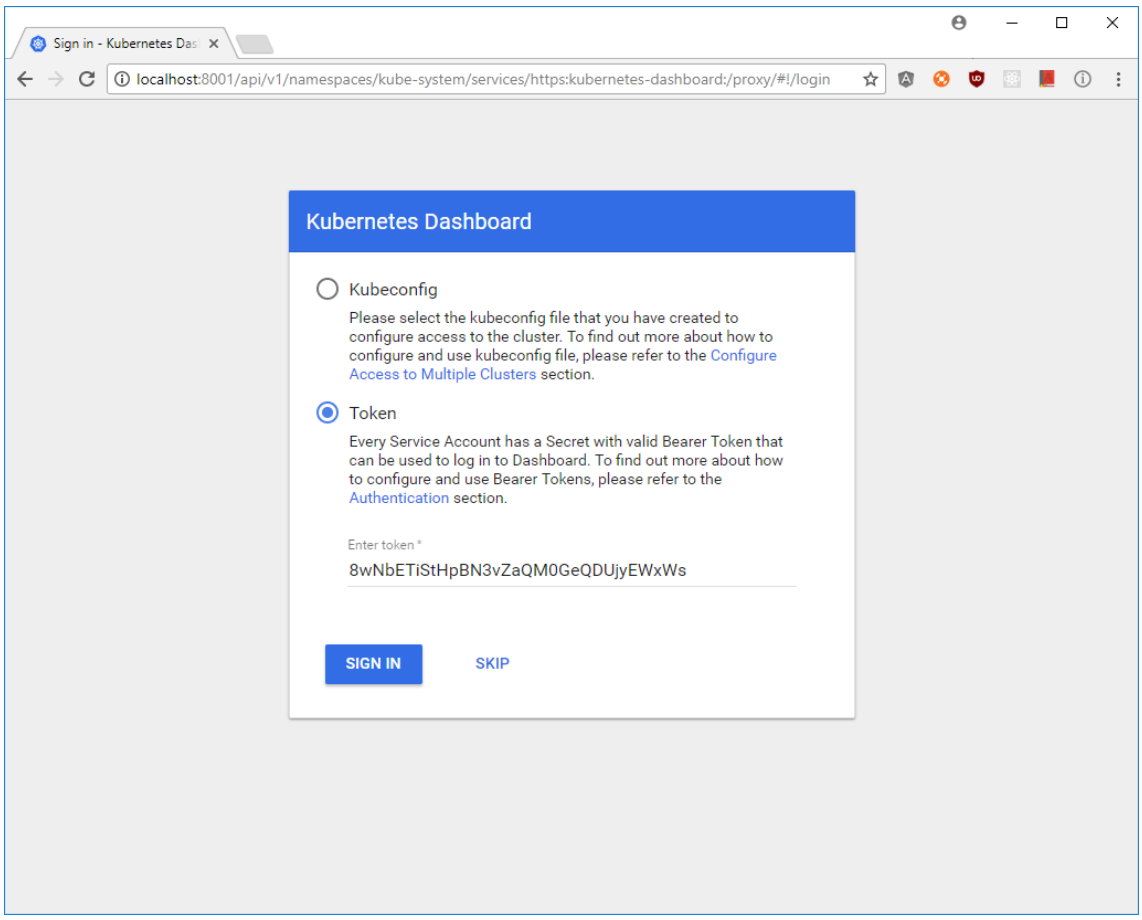
Kubectl-työkalua käyttämällä asennettiin ohjauspaneeli Kubernetes-klusteriin. Asennus tapahtui oheisella komennolla:

```
$ kubectl apply -f https://raw.githubusercontent.com/kuber-
netes/dashboard/master/src/deploy/recommended/kubernetes-dash-
board.yaml
```

Ohjauspaneeliin ei oletusarvoisesti pääse klusterin ulkopuolelta. Avattiin oheisella komennolla välityspalvelin Kubernetes-klusterin rajapintaan projektikoneelta:

```
$ kubectl proxy
```

Komennon jälkeen ohjauspaneeli oli saavutettavissa projektikoneella osoitteessa http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login. Ohjauspaneeliin kirjauduttiin aikaisemmin luodulla staattisella tokenilla kuvan 16 mukaisesti.



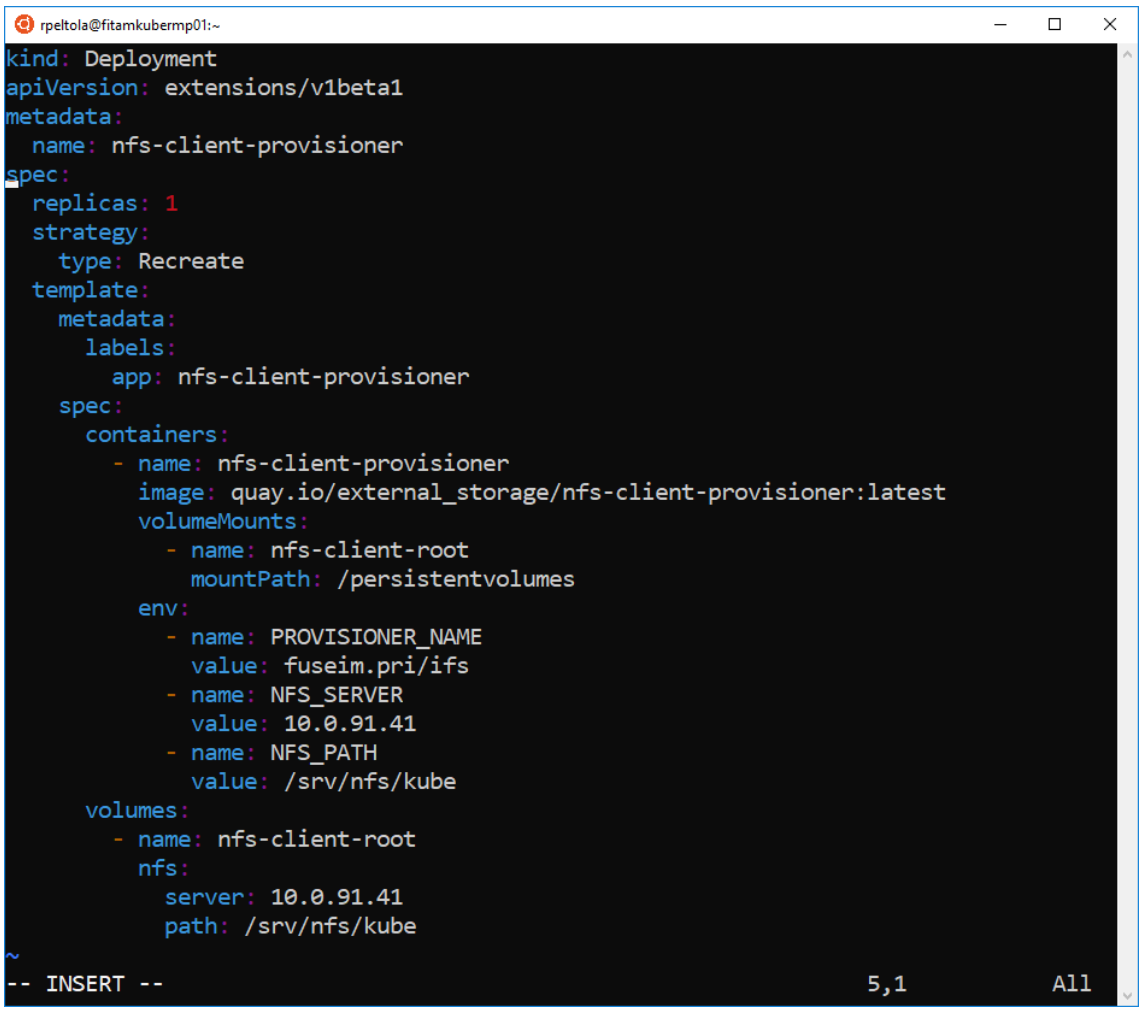
KUVA 16. Ohjauspaneelin kirjautumisnäkymä

4.7 Pysyvän tallennustilan käyttöönotto

Projektia varten yrityksen NFS-palvelimelta saatiin käyttöön tallennustilaa. Tallennustila löytyi palvelimelta, jonka IP-osoite oli 10.0.91.41 ja NFS-jaon hakemisto oli `/srv/nfs/kube`. Koska Kuberneteksessä ei ole sisäänrakennettua tukea NFS-volumeille, niin käytettiin kubernetes-incubator nimisen projektin alla olevaa external-storage kirjastoa, joka tarjosi Kubernetekseen NFS-tuen. (Storage Classes 2017.)

Kuberneteksen master-nodelle ladattiin deployment.yaml-tiedosto oheisella komennolla ja muokattiin sitä kuvan 17 mukaisesti sopimaan projektin ympäristöön.

```
$ wget https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/nfs-client/deploy/deployment.yaml
```



KUVA 17. deployment.yaml-tiedoston muokkaus projektin NFS-palvelimen kanssa yhteensopivaksi

Muokkauksen jälkeen suoritettiin oheinen komento, joka latasi tiedostot Kubernetekseen:

```
$ kubectl create -f deployment.yaml
$ kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/nfs-client/deploy/class.yaml
```

Koska projektin Kuberneteksessä oli käytössä RBAC, eli roolipohjainen valtuutus, tuli myös ladata käyttöön muutama muu konfiguraatio (Kubernetes-incubator: external-storage 2017).

Palvelutunnuksen luomiseen tarvittava konfiguraatio ladattiin järjestelmään oheisella komennolla:

```
$ kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/nfs-client/deploy/auth/serviceaccount.yaml
```

Palvelutunnuksen luomisen jälkeen ladattiin tarvittava rooli-konfiguraatio projektikoneelle, korjattiin siinä oleva API-versio yhteensopivaksi Kuberneteksen 1.8 version kanssa ja ladattiin konfiguraatio Kubernetekseen oheisilla komennoilla:

```
$ wget https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/nfs-client/deploy/auth/clusterrole.yaml
$ sed -i s@rbac.authorization.k8s.io/v1alpha1@rbac.authorization.k8s.io/v1@g clusterrole.yaml
$ kubectl create -f clusterrole.yaml
```

Projektikoneelle ladattiin konfiguraatio, jolla aikaisempi konfiguraatio liitettiin jo luotuun palvelutunnukseen. Korjattiin siinä oleva API-versio yhteensopivaksi Kubernetes 1.8 version kanssa ja ladattiin konfiguraatio järjestelmään oheisilla komennoilla:

```
$ wget https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/nfs-client/deploy/auth/clusterrolebinding.yaml
$ sed -i s@rbac.authorization.k8s.io/v1alpha1@rbac.authorization.k8s.io/v1@g clusterrolebinding.yaml
$ kubectl create -f clusterrolebinding.yaml
```

Muokattiin vielä alkuperäiseen deploymenttiin käyttöön juuri luotu palvelutunnus oheisella komennolla:

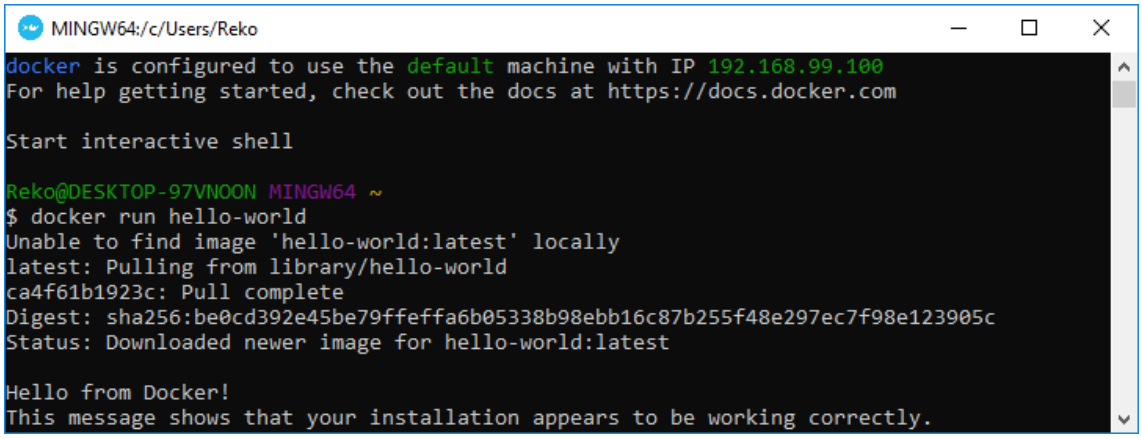
```
$ kubectl patch deployment nfs-client-provisioner -p
'{"spec":{"template":{"spec":{"serviceAccount":"nfs-client-
provisioner"}}}}'
```

5 KÄYTTÖÖNOTTO

5.1 Projektikoneen valmistelu

Projektikoneelle tarvittiin toimiva Docker-asennus Docker registryn sekä kahden projektin dockeroinnin testaamista varten. Tähän käytettiin Docker Toolbox-nimistä projektia, jonka asennusohjeet löytyivät osoitteesta https://docs.docker.com/toolbox/toolbox_install_windows/.

Asennuksen jälkeen käynnistettiin Docker Toolbox klikkaamalla Windowsin käynnistävalikosta Docker Quickstart Terminal nimistä kuvaketta (Install Docker Toolbox on Windows 2017). Ohjelma kysyi lupaa konfiguroida verkkoasetuksia, ja näihin tuli antaa lupa. Hetken kuluttua näytölle tulostui tieto IP-osoitteesta, ja tämän jälkeen testattiin asennuksen onnistuneisuus ajamalla dockerin hello-world -komento kuvan 18 mukaisesti.



KUVA 18. Onnistunut Docker Toolbox-asennus

5.2 Välityspalvelimen asennus

Kuberneteksen palveluita varten asennettiin välityspalvelin, jonka kautta HTTP- ja HTTPS-liikenne ohjattiin Kubernetekseen. Välityspalvelimena toimii CentOS 7 -käyttöjärjestelmän päälle asennettava NGINX WWW-palvelinohjelmisto, joka asennettiin oheisella komennolla:

```
$ sudo yum install -y nginx
```

Asennuksen jälkeen nginx-palvelimeen laitettiin paikalleen SSL-sertifikaatti, jolla HTTPS saatiin toimimaan. Sertifikaatti laitettiin palvelimella `/etc/ssl/certs` -hakemistoon ja sertifikaatin avain `/etc/ssl/private` hakemistoon. Jos sertifikaattia ei ole, se voidaan hankkia palveluntarjoajalta tai ilmaisesta Let's Encrypt -nimisestä palvelusta.

5.3 Docker registryn asennus

Omille Docker imageille tarvittiin Docker registry, johon projekteista luodut imaget laitetaan ja mistä Kubernetes sai haettua ne. Tässä projektissa käytettiin omaa docker-registyä, joka asennettiin Kubernetekseen palveluna. Vaihtoehtoisesti voitaisiin myös käyttää Docker Hub -nimistä palvelua.

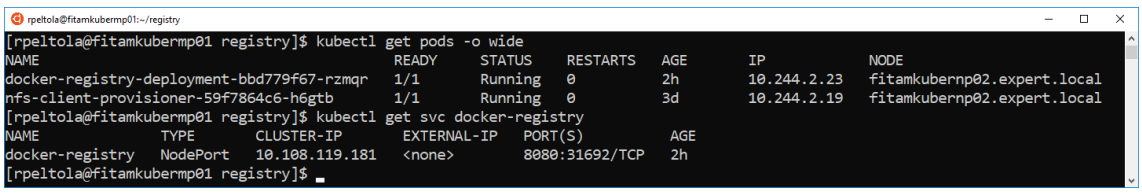
Docker registry tarvitsi pysyvää tallennustilaa Docker imageille. Käytettiin tähän aikaisemmin konfiguroitua NFS-tallennustilaa, jota saatiin käyttöön luomalla uusi persistent volume claim liitteen 3 sisällöstä ja lataamalla se järjestelmään oheisella komennolla:

```
$ kubectl create -f docker-registry-claim.yaml
```

Tallennustilan konfiguroinnin jälkeen asennettiin Kubernetekseen itse Docker registry -ohjelmisto. Liitteessä 4 on luotuna tarvittava Kuberneteksen palvelu sekä deployment, joka asennettiin Kubernetekseen oheisella komennolla:

```
$ kubectl create -f docker-registry.yaml
```

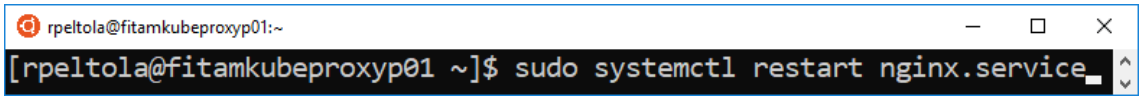
Asennuksen jälkeen tarkistettiin kuvan 19 mukaisesti luodun Docker registry-deployment -podin tila, jonka tuli olla tilassa running. Lisäksi etsittiin luodun Docker registry -nimisen palvelun portti, joka tässä tapauksessa oli 31692.



KUVA 19. Docker registry -podin tilan tarkistus, ja palvelun portin listaaminen

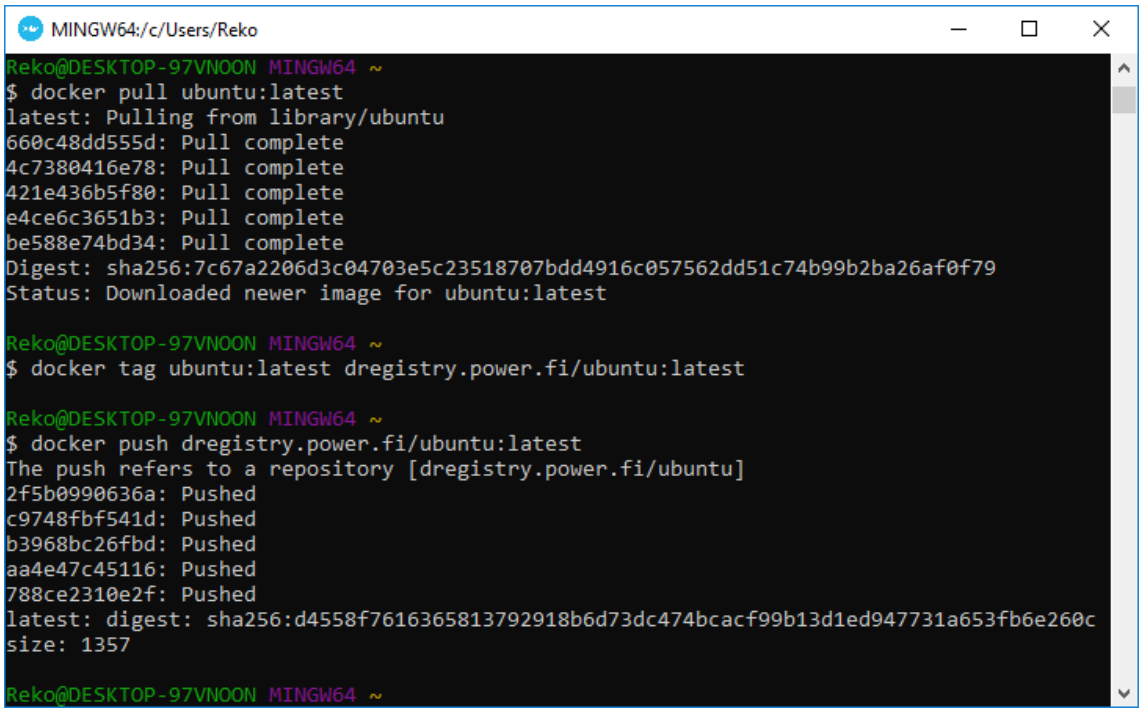
Välityspalvelimeen tuli vielä lisätä konfiguraatio Docker registrylle. Liitteen 5 konfiguraatioon muokattiin oikea polku ssl-sertifikaatille ja palvelimen nimi vaihettiin sopivaksi. Lisäksi proxy_pass rivin IP-osoite ja portti vaihdettiin ympäristöön sopivaksi.

Muokkausten jälkeen konfiguraatio kopioitiin välityspalvelimen `/etc/nginx/nginx.conf` tiedostoon ennen ensimmäistä ”server {” -osuutta ja käynnistettiin välityspalvelin uudelleen kuvan 20 mukaisella komennolla.



KUVA 20. Välityspalvelimen WWW-palvelimen uudelleen käynnistäminen

Projektin Docker registryä voitiin testata hakemalla Ubuntun Docker image ja puskemalla se projektin Docker registryyn kuvan 21 mukaisesti. Docker image voitiin puskea haluttuun Docker registryyn laittamalla sille sopiva tunniste, eli tagi (Docker tag 2017). Tässä tapauksessa haluttu Docker registry oli dregistry.power.fi, joten merkittiin image sen mukaisesti.

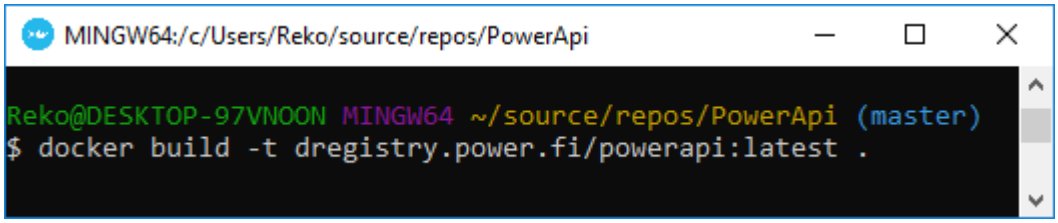


KUVA 21. Ubuntu-imagen puskeminen omaan Docker registryyn

5.4 Myyntisovelluksen dockerointi

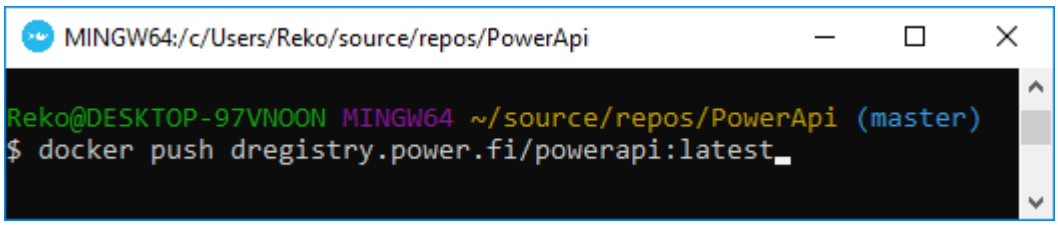
Myyntisovellus koostui React-teknologialla tehdystä selainsovelluksesta, sekä ASP.NET Core 2.0 -tekniikalla tehdystä palvelinpuolesta. Palvelin käytti lisäksi toista ASP.NET Core 1.1 tekniikalla tehtyä sisäistä rajapintaa, joka myös dockeroitiin.

ASP.NET Core-projektien dockerointiin käytettiin Microsoftin tarjoamia Docker imageita ASP.NET Core teknologialle. Microsoftin Docker Hub:ista löytyi hyvä esimerkki Dockerfilen muodostamista varten. Microsoftin esimerkki käytti Dockerin multi-stage buildia, joka tarkoittaa käytännössä sitä, että Docker imagen rakentamiseen käytetään useampaa eri Docker containeria. Ensimmäistä containeria käytettiin ohjelman kääntämiseen, jolloin projektikoneella ei tarvinnut olla projektiin tarvittavia käännöstyökaluja. Lopputuloksena syntyvään Docker imageen kopioitiin ensimmäisessä containerissa valmiiksi käännetty ohjelma. (Docker Hub: microsoft/dotnet 2017; Use multi-stage builds 2017.)



KUVA 22. Komento Docker imagen luomista varten

ASP.NET Core-projekteja varten muodostetut Dockerfilet ovat liitteissä 6 ja 7. Dockerfilet sijoitettiin projektien juurihakemistoihin. Docker imagen luominen tapahtui Docker Toolboxilla kuvan 22 mukaisesti, jossa projektin juurihakemistossa kutsuttiin `docker build` -komentoa. Komennolle annettiin parametrina tag, jossa oli haluttu Docker registry ja projektille sopiva nimi. Lisäksi merkattiin image latest-tunnisteella, joka tarkoittaa uusinta versiota. Lopuksi image puskettiin projektin Docker registryyn kuvan 23 mukaisella komennolla.



KUVA 23. Komento, jolla Docker image pusketaan Docker registryyn

Myyntisovelluksen selainosuutta varten muodostettiin liitteen 8 mukainen Dockerfile. Siinä käytettiin erillistä npm-containeria sovelluksen kääntämiseen ja lopuksi kopioitiin käännetyt tiedostot lopulliseen NGINX WWW-palvelin containeriin.

5.5 Indeksin päivittäjän dockerointi

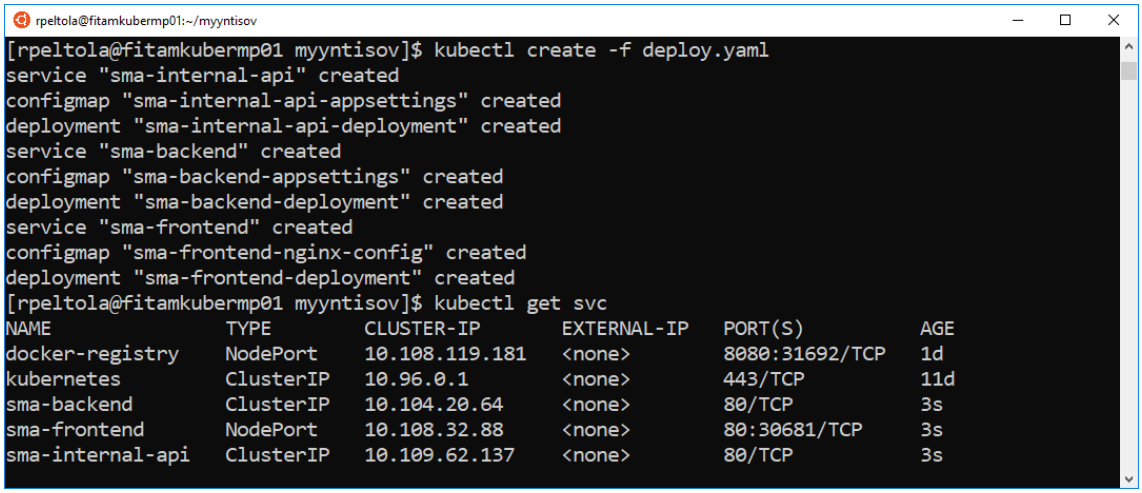
Indeksin päivittäjä on ASP.NET Core 2.0 sovellus, jota ajetaan ajastetusti kerran yössä. Sovelluksen tehtävä on päivittää myyntisovelluksen tarvitsemia tietoja Elasticsearch-palvelimen indeksiin, josta niitä voidaan hakea.

Projektin Dockerfile on esitetty liitteessä 9. Se vastaa projektirakennetta lukuun ottamatta etämyyntisovelluksen Dockerfileä.

5.6 Myyntisovelluksen ajaminen Kuberneteksessa

Myyntisovellus koostui kolmesta eri komponentista, jotka liitettiin yhteen Kuberneteksen service-abstraktiolla. Sovellusten tarvitsemat konfiguraatiot tallennettiin Kuberneteksen config maps -toiminnallisuuteen.

Sovellusta varten luotiin liitteen 10 mukainen konfiguraatio, jossa jokaiselle kolmelle komponentille on luotu deployment ja service. Selainsovellukseen, joka on konfiguraatiossa nimellä sma-frontend, tuli päästä käsiksi Kuberneteksen ulkopuolelta, joten service on tyyppiä NodePort. Muihin palveluihin riitti klusterin sisällä toimiva ClusterIP -service. Selainsovelluksen WWW-palvelimen tarvitsema konfiguraatio luotiin Kuberneteksen config maps -toiminnallisuuteen, ja asetettiin sieltä podille. Muissa palveluissa konfiguraatio tuotiin podeille ympäristömuuttujina, joista ASP.NET Core -sovellukset osasivat lukea ne.

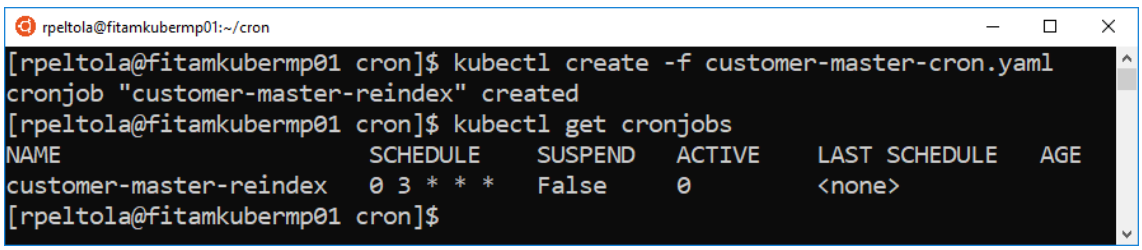


KUVA 24. Myyntisovelluksen asentaminen Kubernetekseen ja palveluiden listaaminen

Sovelluksen konfiguraatio ajettiin Kubernetekseen kuvan 24 mukaisella komennolla. Komennon jälkeen selvitettiin luodun sma-frontend -nimisen palvelun portti välityspalvelinta varten. Liitteen 11 konfiguraatioon muokattiin oikea portti ja konfiguraatio asetettiin välityspalvelimen `nginx.conf` -tiedostoon Docker registryn yläpuolelle. NGINX käynnistettiin vielä uudelleen kuvan 20 mukaisella komennolla, jonka jälkeen myyntisovellus oli saavutettavissa konfiguroidussa osoitteessa.

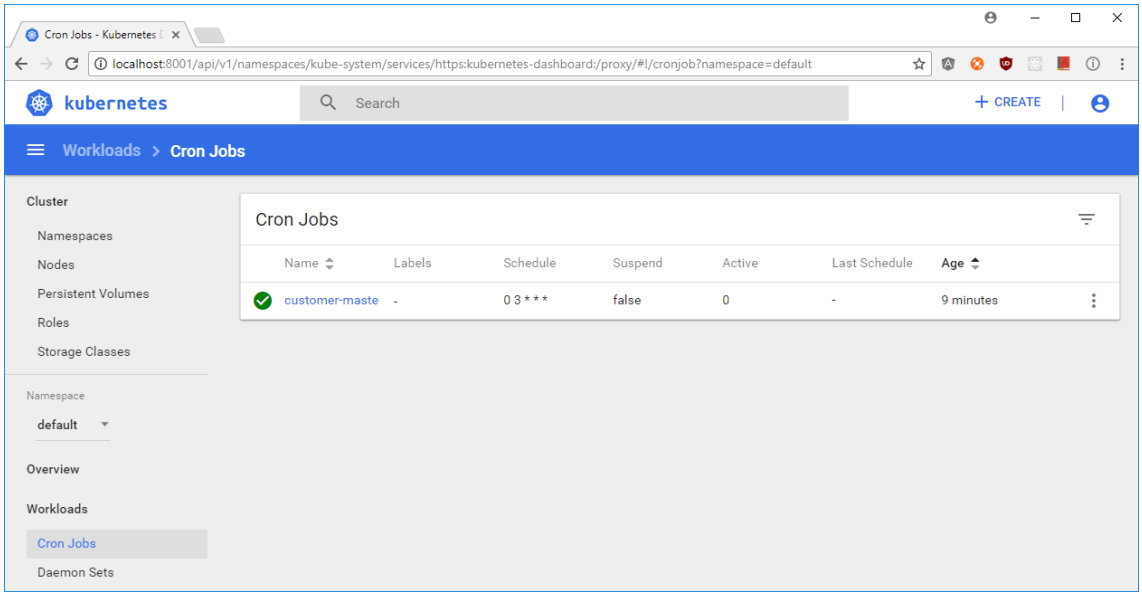
5.7 Indeksin päivittäjän ajaminen Kuberneteksessa

Indeksin päivittäjää varten luotiin liitteen 12 mukainen Cron Job -konfiguraatio Kubernetekseen. Konfiguraatiossa schedule-rivi kertoo cronin syntaksin mukaisesti ajastuksen (Cron 2017). Esimerkki on ajastettu käynnistymään joka yö tasan kello kolme.



KUVA 25. Cron Jobin luominen Kubernetekseen ja niiden listaaminen

Konfiguraatio tallennettiin Kubernetekseen kuvan 25 mukaisella `kubectl create` -komennolla. Kuvassa lisäksi listattiin kaikki Kubernetekseen ajastetut Cron Jobit, josta nähtiin juuri ajastetun tehtävän tiedot. Samat tiedot nähtiin myös ohjauspaneelista kirjautumisen jälkeen, kun kuvan 26 mukaisesti valittiin vasemmasta laidasta Cron Jobs -niminen valikko.



KUVA 26. Cron Jobit listattuna ohjauspaneelissa

6 POHDINTA

Projektin tavoitteena oli asentaa Kubernetes-klusteri ja ottaa se rajoitettuun testikäyttöön muutamalle eri projektille. Projekti kokonaisuudessaan dokumentoitiin tulevaa laajempaa tuotantokäyttöönottoa varten.

Projekti onnistui tavoitteiden mukaisesti, ja käyttöön saatiin Kubernetes-klusteri. Projektin aikana luotiin Dockerfilet kahdelle eri projektille, joiden avulla projekteista saatiin tehtyä Docker imaget. Dockerfilet toimivat myös esimerkkeinä tulevaa käyttöönottoa varten muille samantapaisille projekteille. Projekteille luotiin lisäksi konfiguraatiotiedostot Kubernetesta varten, jolla projektit saatiin ajoon Kubernetekseen. Näitä konfiguraatioita voidaan hyödyntää esimerkkeinä myöhempiä projekteja varten.

Projektin jatkokehitystyönä Kubernetekseen tuodaan tuki Windows-käyttöjärjestelmällä varustetuille nodeille. Kuberneteksen Windows-nodeilla saadaan tuki Docker for Windows -teknologialle, jota tarvitaan Powerin vanhempien projektien siirtämisessä Kubernetekseen.

Powerin vanhempien, Windows-käyttöjärjestelmää vaativien, projektien osalta erityisesti tasks-projekti, jossa on noin 20 erilaista ajastettua tehtävää, hyötyisi Kubernetekseen siirrosta. Projektin päivityksessä on nykyisin erityisiä haasteita, koska eri tehtävät on jaettu eri palvelimille kuorman tasausta varten. Projektin päivitysten tekemiseen kuluu nykyisin runsaasti aikaa ja inhimillisten virheiden mahdollisuus on suuri, koska päivityksiä tehdään usein. Kuberneteksessä riittäisi, että projektin käyttämä Docker image päivitetään, jolloin myös kaikki tehtävät päivittyisivät.

Tässä projektissa käyttäjien kirjautuminen Kubernetekseen hoidettiin staattisilla tokeneilla, mutta niiden ylläpito isommassa mittakaavassa ei ole järkevää, ja niiden muuttaminen vaatii myös Kuberneteksen API-palvelimen uudelleenkäynnistämisen. Tulevaa laajempaa tuotantokäyttöä varten Kubernetes-klusteriin tokenien tilalle olisi hyvä ottaa käyttöön esimerkiksi välityspalvelimen kautta tapahtuva tunnistautuminen, jonka kautta hoidettaisiin myös käyttäjien roolit.

Laajempaa käyttöä varten master-node olisi hyvä myös replikoida, koska kubeadm-komennon luoma klusteri koostuu vain yhdestä master-nodesta. Lisäksi Kuberneteksen toimintaa tulee testata häiriötilanteissa laajasti. Esimerkiksi GitHub omassa testauksessaan huomasi, että Kuberneteksen API-palvelimen noden kaatuminen voi aiheuttaa huomattavia häiriöitä klusterin toimintaan, jonka vuoksi GitHub päätyi käyttämään kahta erillistä Kubernetes-klusteria (Kubernetes at GitHub 2017).

LÄHTEET

About images, containers, and storage drivers. (2017). Luettu 9.12.2017. https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

Authenticating. (2017). Luettu 9.12.2017. https://kubernetes.io/docs/admin/authentication/

Azure Container Service (AKS). (2017). Luettu 10.12.2017. https://azure.microsoft.com/en-us/services/container-service/

Cluster Networking. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/concepts/cluster-administration/networking/

Containerd Brings More Container Runtime Options for Kubernetes. (2.11.2017). Luettu 10.12.2017. http://blog.kubernetes.io/2017/11/containerd-container-runtime-options-kubernetes.html

Cron. (9.12.2017). Luettu 12.12.2017. https://en.wikipedia.org/wiki/Cron

Cron Jobs. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

Docker Basics - Amazon Elastic Container Service. (2017). Luettu 10.12.2017. http://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

Docker For Windows Server. (2017). Luettu 9.12.2017. https://www.docker.com/docker-windows-server

Docker Hub: microsoft/dotnet. (2017). Luettu 11.12.2017. https://hub.docker.com/r/microsoft/dotnet/

Docker overview. (2017). Luettu 9.12.2017. https://docs.docker.com/engine/docker-overview/

Docker Registry. (2017). Luettu 9.12.2017. https://docs.docker.com/registry/

Docker tag. (2017). Luettu 11.12.2017. https://docs.docker.com/engine/reference/commandline/tag/

Dockerfile reference. (2017). Luettu 9.12.2017. https://docs.docker.com/engine/reference/builder/

Flannel is a network fabric for containers. (2017). Luettu 10.12.2017. https://github.com/coreos/flannel

Glider Labs. (2017). Luettu 9.12.2017. https://github.com/gliderlabs/docker-alpine

Install Docker Toolbox on Windows. (2017). Luettu 11.12.2017. https://docs.docker.com/toolbox/toolbox_install_windows/#step-3-verify-your-installation

Installing kubeadm. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/setup/independent/install-kubeadm/

Kubernetes at GitHub. (16.8.2017). Luettu 13.12.2017. https://githubengineering.com/kubernetes-at-github/

Kubernetes engine. (2017). Luettu 10.12.2017. https://cloud.google.com/kubernetes-engine/

Kubernetes: Finally...A True Cloud Platform (2016). Katsottu 9.12.2017. https://www.youtube.com/watch?v=of45hYbkIZs

Kubernetes-incubator: external-storage. (2017). Luettu 9.12.2017. https://github.com/kubernetes-incubator/external-storage/tree/master/nfs-client

Pods. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/concepts/workloads/pods/pod/

Replication Controller. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/

Services. (2017). Luettu 12.12.2017 https://kubernetes.io/docs/concepts/services-networking/service/

Storage Classes. (2017). Luettu 9.12.2017. https://kubernetes.io/docs/concepts/storage/storage-classes/

Use multi-stage builds. (2017). Luettu 11.12.2017. https://docs.docker.com/engine/userguide/eng-image/multistage-build/

Using RBAC Authorization. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/admin/authorization/rbac/

Volumes. (2017). Luettu 10.12.2017. https://kubernetes.io/docs/concepts/storage/volumes/

Windows 10 Installation Guide. (11.7.2017). Luettu 9.12.2017. https://docs.microsoft.com/en-us/windows/wsl/install-win10

LIITTEET

Liite 1. CentOS 7 pakettihakemisto kuberneteksen paketeille

```
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-
x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
```

Liite 2. kube-apiserver.yaml konfiguraatiotiedosto

1 (2)

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/critical-pod: ""
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-
kubelet-client.crt
    - --enable-bootstrap-token-auth=true
    - --requestheader-group-headers=X-Remote-Group
    - --service-cluster-ip-range=10.96.0.0/12
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --tls-cert-file=/etc/kubernetes/pki/apiserver.crt
    - --secure-port=6443
    - --kubelet-preferred-address-
types=InternalIP,ExternalIP,Hostname
    - --requestheader-username-headers=X-Remote-User
    - --requestheader-allowed-names=front-proxy-client
    - --service-account-key-file=/etc/kubernetes/pki/sa.pub
    - --admission-
control=Initializers,NamespaceLifecycle,LimitRanger,ServiceAccount,
PersistentVolumeLabel,DefaultStorageClass,DefaultTolerationSeconds,
NodeRestriction,ResourceQuota
    - --allow-privileged=true
    - --requestheader-extra-headers-prefix=X-Remote-Extra-
    - --advertise-address=10.0.91.40
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-
client.key
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-
proxy-ca.crt
    - --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-
client.crt
    - --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-
client.key
    - --insecure-port=0
    - --authorization-mode=Node,RBAC
    - --etcd-servers=http://127.0.0.1:2379
    - --token-auth-file=/etc/kubernetes/static-token.csv
    image: gcr.io/google_containers/kube-apiserver-amd64:v1.8.4
    livenessProbe:
      failureThreshold: 8
      httpGet:
        host: 127.0.0.1
        path: /healthz
```

(jatkuu)

2 (2)

```
        port: 6443
        scheme: HTTPS
      initialDelaySeconds: 15
      timeoutSeconds: 15
    name: kube-apiserver
    resources:
      requests:
        cpu: 250m
    volumeMounts:
    - mountPath: /etc/kubernetes/pki
      name: k8s-certs
      readOnly: true
    - mountPath: /etc/ssl/certs
      name: ca-certs
      readOnly: true
    - mountPath: /etc/pki
      name: ca-certs-etc-pki
      readOnly: true
    - mountPath: /etc/kubernetes/static-token.csv
      name: static-token-csv-file
      readOnly: true
  hostNetwork: true
  volumes:
  - hostPath:
      path: /etc/kubernetes/pki
      type: DirectoryOrCreate
    name: k8s-certs
  - hostPath:
      path: /etc/ssl/certs
      type: DirectoryOrCreate
    name: ca-certs
  - hostPath:
      path: /etc/pki
      type: DirectoryOrCreate
    name: ca-certs-etc-pki
  - hostPath:
      path: /etc/kubernetes/static-token.csv
      type: FileOrCreate
    name: static-token-csv-file
status: {}
```

Liite 3. docker-registry-claim.yaml -konfiguraatiotiedosto

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: docker-registry-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: "managed-nfs-storage"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Liite 4. docker-registry.yaml -konfiguraatiotiedosto

```
apiVersion: v1
kind: Service
metadata:
  name: docker-registry
  labels:
    app: docker-registry
spec:
  type: NodePort
  ports:
  - port: 8080
    targetPort: 5000
    protocol: TCP
    name: registry-http
  selector:
    app: docker-registry
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: docker-registry-deployment
spec:
  selector:
    matchLabels:
      app: docker-registry
  replicas: 1
  template:
    metadata:
      labels:
        app: docker-registry
    spec:
      containers:
      - name: docker-registry
        image: registry:2
        ports:
        - containerPort: 5000
        volumeMounts:
        - name: docker-registry-storage
          mountPath: "/var/lib/registry"
      volumes:
        - name: docker-registry-storage
          persistentVolumeClaim:
            claimName: docker-registry-claim
```

Liite 5. docker-registryn konfiguraatio välityspalvelimelle

```
    # Settings for a TLS dregistry.power.fi
    server {
        client_max_body_size 1G;
        listen       443 ssl http2;
        listen       [::]:443 ssl http2;
        server_name  dregistry.power.fi;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/ssl/certs/wildcard_power_fi.crt";
        ssl_certificate_key
"/etc/ssl/private/wildcard_power_fi.key";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers on;

        location /
        {
                proxy_set_header  Host              $http_host;   #
required for docker client's sake
                proxy_set_header  X-Real-IP         $remote_addr; #
pass on real client's IP
                proxy_set_header  X-Forwarded-For
$proxy_add_x_forwarded_for;
                proxy_set_header  X-Forwarded-Proto $scheme;
                proxy_read_timeout                  900;
                proxy_pass
http://10.0.91.40:31692;
        }

        error_page 404 /404.html;
            location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
            location = /50x.html {
        }
    }
```

Liite 6. Etämyyntisovelluksen palvelinosuuden Dockerfile

```
FROM microsoft/dotnet:2.0-sdk AS build-env
WORKDIR /app

# copy csprojs and restore as distinct layers
RUN mkdir PowerSMA.Website PowerSMA.Data PowerSMA.Business
COPY ./PowerSMA.Website/*.csproj ./PowerSMA.Website/
COPY ./PowerSMA.Business/*.csproj ./PowerSMA.Business/
COPY ./PowerSMA.Data/*.csproj ./PowerSMA.Data/
COPY ./PowerSMA.sln ./
RUN dotnet restore

# copy everything else and build
COPY ./PowerSMA.Website/* ./PowerSMA.Website/
COPY ./PowerSMA.Data/* ./PowerSMA.Data/
COPY ./PowerSMA.Business/* ./PowerSMA.Business/
RUN cd PowerSMA.Website && \
        # Remove environment specific appsettings from docker-image
    rm appsettings.Development.json appsettings.Test.json
appsettings.Production.json && \
        dotnet publish -c Release -o ../out && \
        # Remove debug-files
        rm ../out/*.pdb

# build runtime image
FROM microsoft/dotnet:2.0-runtime
WORKDIR /app
COPY --from=build-env /app/out ./
ENTRYPOINT ["dotnet", "PowerSMA.Website.dll"]
```

Liite 7. Etämyyntisovelluksen tarvitseman rajapinnan Dockerfile

```
FROM microsoft/dotnet:1.1-sdk AS build-env
WORKDIR /app

# copy csprojs and restore as distinct layers
RUN mkdir Business BusinessModels Data InternalApi
COPY ./BusinessModels/*.csproj ./BusinessModels/
COPY ./Business/*.csproj ./Business/
COPY ./Data/*.csproj ./Data/
COPY ./InternalApi/*.csproj ./InternalApi/
COPY ./PowerApi.sln ./
RUN dotnet restore

# copy everything else and build
COPY ./BusinessModels/* ./BusinessModels/
COPY ./Business/* ./Business/
COPY ./Data/* ./Data/
COPY ./InternalApi/* ./InternalApi/
RUN cd InternalApi && \
        # Remove environment specific appsettings from docker-image
    rm appsettings.Development.json appsettings.Test.json
appsettings.Production.json && \
        dotnet publish -c Release -o ../out && \
        # Remove debug-files
        rm ../out/*.pdb

# build runtime image
FROM microsoft/dotnet:1.1-runtime
WORKDIR /app
COPY --from=build-env /app/out ./
ENTRYPOINT ["dotnet", "InternalApi.dll"]
```

Liite 8. Etämyyntisovelluksen selainsovelluksen Dockerfile

```
FROM node:9-stretch AS build-env
WORKDIR /app

# package.json for fetching deps
COPY package.json .
COPY package-lock.json .
RUN npm install

# Copy all files and build
COPY . .
RUN npm run bundle

# build runtime image

# slim alpine based nginx
FROM nginx:stable-alpine

# copy static html-files for nginx
COPY --from=build-env /app/index.html /usr/share/nginx/html/
COPY --from=build-env /app/build/ /usr/share/nginx/html/build/
```

Liite 9. Indeksin päivittäjän Dockerfile

```
FROM microsoft/dotnet:2.0-sdk AS build-env
WORKDIR /app

# copy csprojs and restore as distinct layers
RUN mkdir CustomerMaster.Business CustomerMaster.Database
CustomerMaster.Elastic CustomerMaster.Library CustomerMaster.Tasks
COPY ./CustomerMaster.Business/*.csproj ./CustomerMaster.Business/
COPY ./CustomerMaster.Database/*.csproj ./CustomerMaster.Database/
COPY ./CustomerMaster.Elastic/*.csproj ./CustomerMaster.Elastic/
COPY ./CustomerMaster.Library/*.csproj ./CustomerMaster.Library/
COPY ./CustomerMaster.Tasks/*.csproj ./CustomerMaster.Tasks/
COPY ./CustomerMaster.sln ./
RUN dotnet restore

# copy everything else and build
COPY ./CustomerMaster.Business/ ./CustomerMaster.Business/
COPY ./CustomerMaster.Database/ ./CustomerMaster.Database/
COPY ./CustomerMaster.Elastic/ ./CustomerMaster.Elastic/
COPY ./CustomerMaster.Library/ ./CustomerMaster.Library/
COPY ./CustomerMaster.Tasks/ ./CustomerMaster.Tasks/

RUN cd CustomerMaster.Tasks && \
       dotnet publish -c Release -o ../out && \
       # Remove debug-files
       rm ../out/*.pdb

# build runtime image
FROM microsoft/dotnet:2.0-runtime
WORKDIR /app

COPY --from=build-env /app/out ./
COPY --from=build-env /app/CustomerMaster.Tasks/appsettings.json ./

ENTRYPOINT ["dotnet", "CustomerMaster.Tasks.dll"]
```

Liite 10. Etämyyntisovelluksen konfiguraatio Kubernetekseen

1 (4)

```
# START sma-internal-api
apiVersion: v1
kind: Service
metadata:
  name: sma-internal-api
  labels:
    app: sma-internal-api
    environment: prod
spec:
# External access is not needed
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: sma-internal-api-http
  selector:
    app: sma-internal-api
    environment: prod
---
# Appsettings
apiVersion: v1
kind: ConfigMap
metadata:
  name: sma-internal-api-appsettings
data:
  ASPNETCORE_ENVIRONMENT: Production
  ASPNETCORE_URLS: http://0.0.0.0:80
  ConnectionStrings__umbracoDbDSN: "server=xxx;database=xxx;user
id=xxx;password=xxx"
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: sma-internal-api-deployment
#  environment: prod
spec:
  selector:
    matchLabels:
      app: sma-internal-api
      environment: prod
# this is simple stateless api and can be replicated
  replicas: 2
  template:
    metadata:
      labels:
        app: sma-internal-api
        environment: prod
    spec:
      containers:
      - name: sma-internal-api
        image: dregistry.power.fi/powerapi:latest
        ports:
        - containerPort: 80
        envFrom:
```

(jatkuu)

```
        - configMapRef:
            name: sma-internal-api-appsettings
# END sma-internal-api
---
# START sma-backend
apiVersion: v1
kind: Service
metadata:
  name: sma-backend
  labels:
    app: sma-backend
    environment: prod
spec:
# External access is not needed
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: sma-backend-http
  selector:
    app: sma-backend
    environment: prod
---
# Appsettings
apiVersion: v1
kind: ConfigMap
metadata:
  name: sma-backend-appsettings
data:
  ASPNETCORE_ENVIRONMENT: Production
  ASPNETCORE_URLS: http://0.0.0.0:80
  ConnectionStrings__DbPowerSMA: "server=xxx;database=xxx;user
id=xxx;password=xxx"
  ConnectionStrings__ElasticCustomerServer: "xxx"
# kube-dns will resolve sma-internal-api to correct internal ip-
address
  ConnectionStrings__ApiHost: "http://sma-internal-api"
  ConnectionStrings__WebHost: "https://www.power.fi"
  SystemSettings__AppUrl: "sma-docker.power.fi"
  SystemSettings__AppProtocol: https
  ElasticSettings__ElasticServerDefaultIndex: "index"
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: sma-backend-deployment
#  environment: prod
spec:
  selector:
    matchLabels:
      app: sma-backend
      environment: prod
# this is not stateless and will require some work for replication
(shared redis or similar)
  replicas: 1
  template:
    metadata:
      labels:
```

```
        app: sma-backend
        environment: prod
    spec:
      containers:
      - name: sma-backend
        image: dregistry.power.fi/powersma-backend:latest
        ports:
        - containerPort: 80
        envFrom:
        - configMapRef:
            name: sma-backend-appsettings
# END sma-backend
---
# START sma-frontend
apiVersion: v1
kind: Service
metadata:
  name: sma-frontend
  labels:
    app: sma-frontend
    environment: prod
spec:
# we need external access for this
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: sma-frontend-http
  selector:
    app: sma-frontend
    environment: prod
---
# Appsettings
apiVersion: v1
kind: ConfigMap
metadata:
  name: sma-frontend-nginx-config
data:
  default.conf: |
    server {
      listen                  80;
      server_name             _;
      root                    /usr/share/nginx/html;
      # everyhing to /
      location / {
        try_files $uri /index.html;
      }
      location /api {
        # kube-dns will resolve this
        proxy_pass http://sma-backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
      }
      # this is needed for auth-redirect
      location /auth-hook {
```

```
        # kube-dns will resolve this
        proxy_pass http://sma-backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection keep-alive;
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
      }
    }
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: sma-frontend-deployment
#  environment: prod
spec:
  selector:
    matchLabels:
      app: sma-frontend
      environment: prod
# frontend is also stateless and can be replicated by default
  replicas: 2
  template:
    metadata:
      labels:
        app: sma-frontend
        environment: prod
    spec:
      containers:
      - name: sma-frontend
        image: dregistry.power.fi/powersma-frontend:latest
        ports:
        - containerPort: 80
        volumeMounts:
        - name: sma-frontend-nginx-config-volume
          mountPath: /etc/nginx/conf.d
      # Load the configuration files for nginx
      volumes:
        - name: sma-frontend-nginx-config-volume
          configMap:
            name: sma-frontend-nginx-config
# END sma-frontend
---
```

Liite 11. Etämyyntisovelluksen konfiguraatio välityspalvelimelle

```
    # sma-docker
    server {
        client_max_body_size 1G;
        listen       443 ssl http2;
        listen       [::]:443 ssl http2;
        server_name  sma-docker.power.fi;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/ssl/certs/wildcard_power_fi.crt";
        ssl_certificate_key
"/etc/ssl/private/wildcard_power_fi.key";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers on;

        location /
        {
                proxy_set_header  Host              $http_host;   #
required for docker client's sake
                proxy_set_header  X-Real-IP         $remote_addr; #
pass on real client's IP
                proxy_set_header  X-Forwarded-For
$proxy_add_x_forwarded_for;
                proxy_set_header  X-Forwarded-Proto $scheme;
                proxy_read_timeout                  900;
                proxy_pass
http://10.0.91.40:30681;
        }

        error_page 404 /404.html;
            location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
            location = /50x.html {
        }
    }
```

Liite 12. Kuberneteksen Cron Job -konfiguraatio indeksin päivittäjälle

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: customer-master-reindex
spec:
  schedule: "0 3 * * *"
  concurrencyPolicy: Forbid
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: customer-master-reindex
            image: dregistry.power.fi/customermaster-tasks:latest
          restartPolicy: OnFailure
```